VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Analýza distribuovaných algoritmů pomocí Petriho sítí

Analysis of Distributed Algorithms by Means of Petri Nets

2011 Tomáš Jankovský

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava*.

V Ostravě 2011 .....

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 2011 .....

Rád bych na tomto místě poděkoval panu profesorui Petru Jančarovi, bez jehož vedení a cenných rad by tato práce nemohla vzniknout.

Abstrakt

Tato diplomová práce popisuje možnosti modelování a analýzy distribuovaných algoritmů pomocí Petriho sítí. Algoritmy typicky představené pomocí pseudokódu, jsou pomocí popsaných metod převedeny na Petriho sítě. Práce je doplněna řadou skriptů a animací, vytvořených ve volně dostupných programových prostředcích, pro další přiblížení čtenáři. Tato diplomová práce obsahuje shrnutí a porovnání monografií Elements of Distributed Algorithms od W. Reisiga s monografií Coloured Petri nets od K. Jensena a L. M. Kristensena.

Klíčová slova: Distribuované algoritmy, Petriho sítě, analýza, diplomová práce

Abstract

This diploma thesis describes possibilities of modeling and analysis of distributed algorithms using Petri nets. Algorithms, typically introduced by pseudocode are converted to Petri nets using described methods. Thesis is extended by scripts and animations, created in freely available software, for further approach to reader. This diploma thesis contains summary and comparison between Elements of Distributed Algorithms by W. Reisig and Coloured Petri nets by K. Jensen and L. M. Kristensen.

Keywords: Distributed algorithms, Petri Nets, Analysis, Master thesis

Obsah

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Jak se v současné době rozšiřuje pokrytí oborů lidské činnosti automatickými systémy, tím také narůstá potřeba tyto systémy korektně řídit, zajistit jejich optimální spolupráci a součinnost. Součástí těchto řídících prvků jsou také takzvané distribuované algoritmy. Termín distribuovaný algoritmus se zpočátku vztahoval pouze na systémy, které byly vytvořeny pro práci s mnoha procesory rozmístěnými ve velké geografické oblasti.

V průběhu času se tento termín přenesl a nyní popisuje i algoritmy pracujícími v místní síti nebo případně ve sdílené paměti multiprocesoru. Distribuované algoritmy jsou dnes součástí celé řady aplikací, zahrnující telekomunikace, distribuované informační procesy nebo sledování procesů v reálném čase. Většina současných kontrolních systémů, jako jsou systémy řízení letového provozu nebo systémy kontrolující jaderné elektrárny, jsou kriticky závislé na distribuovaných systémech. Proto je nutné, aby tyto systémy byly maximálně spolehlivé a efektivní, a to i v případě velmi složitých procesů.

Tato diplomová práce se zabývá problematikou základních distribuovaných systémů, jejich reprezentací pomocí Petriho sítí a analýzou, kterou tato reprezentace umožňuje.

1.1 Cíle diplomové práce

Cílem této diplomové práce je seznámit čtenáře s několika vybranými problémy distribuovaných algoritmů a jejich obvyklou reprezentací pomocí programového pseudokódu. Algoritmy obsažené v této diplomové práci jsem studoval z jejich původních zdrojů, a jako jeden z přínosů této diplomové práce vidím seznámení čtenáře s těmito díly.

Programový pseudokód vybraných algoritmů bude porovnán s reprezentací pomocí Petriho sítí. Čtenář se tak může seznámit se základními prvky teorie Petriho sítí a bude schopen porozumět, navrhnout a analyzovat jednoduché algoritmy sám.

Reprezentace distribuovaných algoritmů pomocí Petriho sítí je popsána v monografii Reisig, W. : Elements of distributed algorithms (Springer 1998). Čtenář se bude moci seznámit s obsahem tohoto díla. Další přínos této diplomové práce nabízí porovnání výše uvedené monografie s monografií Coloured Petri nets (Springer 2009), kterou jsem prostudoval až v průběhu vytváření animací, a která na problematiku využití Petriho sítí nahlíží z jiného úhlu.

Důležitou součástí mé diplomové práce jsou elektronické animace popisující distribuované algoritmy, vybrané po dohodě s vedoucím diplomové práce. Algoritmy byly vybrány tak, aby pokryly nejzákladnější problémy v oblasti distribuované informatiky. Z důvodu srovnání jsou uvedené algoritmy také obsaženy v monografii Elements of distributed algorithms. Možnosti modelování a analýzy budou předvedeny na algoritmu vzájemného vyloučení v několika variantách, ve kterých se pokusím zhodnotit využití výpočetní síly Petriho sítí k tomuto účelu. Spolu s elektronickými animacemi vytvořím ve volně dostupných aplikacích soubor příkladů distribuovaných algoritmů, popisovaných v této diplomové práci, které detailněji přiblíží problematiku budoucím diplomantům. Závěrem provedu i srovnání mezi použitými programovými nástroji.

1.2 Shrnutí obsahu jednotlivých kapitol

Teoretická část této diplomové práce se skládá ze čtyř hlavních částí.

Kapitola 2 se zabývá základy distribuovaných algoritmů. Čtenář se seznámí s několika vybranými distribuovanými algoritmy a jejich reprezentací pomocí fragmentů pseudokódů. Na těchto algoritmech bude představeno i použití Petriho sítí jako alternativy pro modelování chování a vlastní analýzu distribuovaných systémů. Pro čtenáře, kteří se s Petriho sítěmi dosud nesetkali, tato kapitola přináší i formální základy teorie Petriho sítí.

Kapitola 3 je zaměřena na algoritmus vzájemného vyloučení. V této kapitole je znázorněno postupné modelování tohoto algoritmu společně s analýzou jednotlivých kroků. Jednoduchý model je podroben analýze a na základě zjištěných nedostatků je k němu připojována další konstrukce, která zajistí všechny požadované vlastnosti, a celý tento proces je několikrát zopakován. Na tomto konkrétním případu bude moci čtenář porovnat rozdíly modelování pomocí pseudokódu a s pomocí Petriho sítí.

Následující kapitola 4 popisuje formální analýzu algoritmů z předešlé kapitoly s ohledem na žádoucí vlastnosti. V rámci této kapitoly se čtenář seznámí se dvěma základními přístupy analýzy Petriho sítí a to s algebraickou metodou pomocí incidenční matice, a dále s metodou grafovou, která pro analýzu využívá speciální struktury obsažené v Petriho síti.

Kapitola 5 přináší čtenáři náhled do monografie Reisig, W.: Elements of Distributed Algorithms (Springer 1998). Čtenář se seznámí se zaměřením této knihy, s použitými technikami jejího autora a v neposlední řadě se srovnáním některých jeho algoritmů s algoritmy, které vzešly z této diplomové práce. Touto kapitolou končí teoretická část diplomové práce.

Kapitola 6 se věnuje vlastnímu postupu vytvoření animací pro tuto diplomovou práci a jejich krátkému popisu.

2 Distribuované algoritmy

2.1 Úvod

Tato kapitola je zaměřena na popis problému distribuovaných algoritmů. Čtenář se seznámí s několika distribuovanými algoritmy, které se používají pro řešení základních výpočetních problémů. S ohledem na možnosti srovnání jsem vybíral pouze takové algoritmy, které jsou obsaženy i v monografii Elements of Distributed Algorithms. Jednotlivé distribuované algoritmy jsou reprezentovány jak v programovém pseudokódu, tak i Petriho sítěmi, které jsou doplněny odpovídajícím teoretickým základem. Informace k této kapitole jsem čerpal z monografií [1], [2] a [3].

2.2 Distribuované algoritmy

V současné době jsme obklopeni množstvím prvků a procesů spojených navzájem v sítích a komunikujících mezi sebou za účelem splnění společného úkolu. Tyto procesy musí jednat spontánně a nezávisle na sobě, ale s ohledem na ostatní, a tak vytvářejí společné - distribuované výpočetní prostředí. I když jednotlivé procesy tohoto prostředí jsou schopné nezávislé práce, pouze jejich spojení umožňuje splnění zadaného úkolu.

Abychom tyto společné úkoly mohli vyřešit, musíme vytvořit sadu pravidel určujících, kdo má co dělat, pokud možno bez synchronizace a dohledu zvenčí. Musíme tedy vytvořit distribuovaný algoritmus, který zaručuje jak korektnost (tedy správné vyřešení zadaného problému) tak i efektivnost (vytvořený algoritmus musí mít přiměřeně malé náklady).

Distribuované algoritmy jsou tedy celky, spojující algoritmy jednotlivých procesů. Tyto procesy jsou často prováděny souběžně na nezávislých procesorech a mají pouze částečnou informaci o tom, co dělá zbytek algoritmu.

Proto základním cílem při tvorbě distribuovaného algoritmu je úspěšná spolupráce jeho jednotlivých částí nezávisle na možných chybách či přerušení komunikace. Jen tyto algoritmy mohou být úspěšně implementovány pro řízení kritických procesů, jako jsou systémy řízení letového provozu nebo systémy řízení jaderných elektráren. Výběr odpovídajícího algoritmu potom závisí na charakteristice problému a také na charakteristice samotného algoritmu.

2.3 Producent - konzument

Jedním z nejjednodušších distribuovaných algoritmů je systém producent - konzument. Můžeme si ho představit jako soustavu dvou procesů, producenta a konzumenta, které vykonávají několik akcí - vyrob, dodej na sklad, vyjmi ze skladu, spotřebuj. Dále budeme uvažovat o skladovém místě s kapacitou jedné položky. Tento algoritmus běžně nachází uplatnění v různých oblastech, od databází, přes komunikační protokoly, až po operační systémy.

Psáno v programovém pseudokódu vypadá tento algoritmus takto:

```
(sklad : integer; sklad := 0);
begin
   parbegin
      producent:  begin P1:   vyrob;
                        if  sklad = 0 then sklad := 1; goto P1
                  end;
      konzument: begin K1:   if  sklad = 1 then sklad := 0; spotřebuj;
                        goto K1;
                  end;
   parend
end;
```

Výpis 1: Producent - konzument

Pseudopříkazy **parbegin** a **parend** vymezují oblast, ve které oba procesy, producent i konzument, probíhají paralelně.

Další možností, jak tento systém popsat, je pomocí reprezentace stavů jednotlivých procesů. Pro tento popis si celý systém rozdělíme na tři podsystémy - producent, sklad a konzument. Každý z těchto podsystémů může nabýt pouze dvou lokálních stavů. Producent se může nacházet buď ve stavu *připraven k výrobě* (pv) nebo *připraven k dodání* na sklad (pd). Konzument má stavy *připraven k odebrání* ze skladu (po) a *připraven ke spotřebě* (ps) a sklad má stavy *plný* (s1) a *prázdný* (s0).

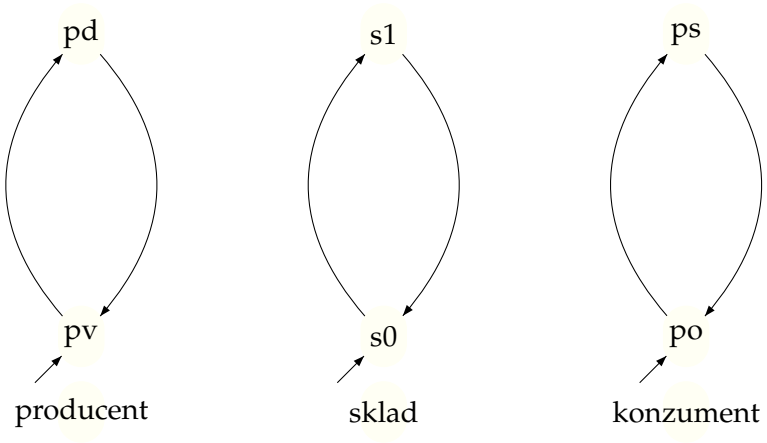


Figure 1: Stavové zobrazení systému producent - konzument

Nicméně tento diagram nepopisuje systém jako celek, a je proto nutné dodat informaci o tom, jak se chovají jednotlivé procesy. Proces $pd \rightarrow pv$ a proces $s0 \rightarrow s1$ probíhají současně. To stejné platí pro procesy $po \rightarrow ps$ a $s1 \rightarrow s0$. Tato informace musí být poskytnuta dodatečně.

Jiný pohled na tento algoritmus představuje popis akcí, které jednotlivé podsystémy provádějí. Producent střídá akci *výroba* (v) a *dodání* na sklad (d) a nekonečná sekvence vdvdvd... popisuje jeho chování. To stejné platí pro akce konzumenta - *odběr* ze skladu (o) a *spotřeba* (s). Akce skladu jsou spojeny s *dodáním* a *odběrem*, označíme je proto $\bar{\mathbf{d}}$ a $\bar{\mathbf{o}}$. Všechny tyto sekvence můžeme zapsat pomocí rovnice:

$$\begin{aligned} \text{producent} &= \text{v.d.producent} \\ \text{konzument} &= \text{o.s.konzument} \\ \text{sklad} &= \bar{\text{d}}.\bar{\text{o}}.\text{sklad} \end{aligned}$$

Celý systém pak můžeme zapsat pomocí paralelního operátoru $\|$, kde pro každé x a $\bar{\text{x}}$ platí, že proběhnou současně.

$$\text{producent} \,\|\, \text{sklad} \,\|\, \text{konzument}$$

Obě výše zmíněné metody popisu systému se dívají na celek jen pod jedním úhlem, buď jako soustavu různých stavů, nebo jako soustavu různých akcí, doplněnou o další potřebné informace.

Na druhé straně stojí metoda popisu pomocí Petriho sítí, která sama o sobě poskytuje výše zmíněné informace, a navíc umožňuje získat informace další.

Na následujícím obrázku stejný systém reprezentovaný Petriho sítí.

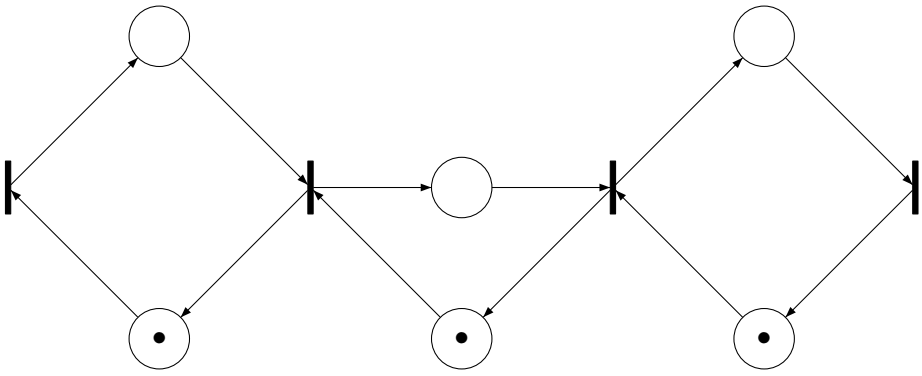


Figure 2: Systém producent - konzument

Z tohoto obrázku, i bez předchozí znalosti Petriho sítí, je možné intuitivně pochopit tento algoritmus i jeho průběh. Levá část je systém producent, ve středu se nalézá sklad a vpravo je systém konzument.

2.4 Definice Petriho sítě, její struktura, přechod

Na příkladu algoritmu producent konzument si definujeme základní prvky Petriho sítí. Petriho sítě se skládají z míst (stavů), které vyjadřují podmínku, a událostí, které jsou reprezentované přechody. Místa jsou graficky znázorněna kružnicí, přechody označujeme obdélníkem, nebo úsečkou. Místa jsou spojena s přechody pomocí orientovaných hran. Vždy je spojeno místo s přechodem nebo přechod s místem. Nikdy nejsou spojeny přechody s jiným přechodem a místo s jiným místem. Výsledkem je bipartitní graf s dvěma typy uzlů, místy a přechody, a s orientovanými hranami. Na následujícím obrázku číslo 3 je jednoduchá síť, která může reprezentovat podsystém producent.

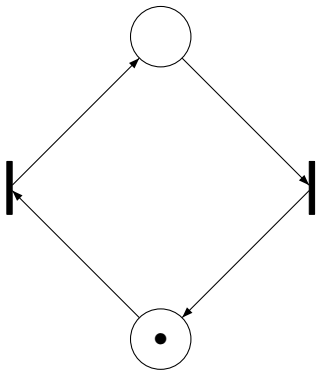


Figure 3: Jednoduchá síť

Nyní si definujeme formálně Petriho síť.

Definice 2.1 *Struktura Petriho sítě je pětice* $\langle P, T, I, O, H \rangle$*, kde*

- *P je konečná množina míst*
- *T je konečná množina přechodů*
- *I,O,H jsou zobrazení typu* $T \rightarrow P^{MS}$*, po řadě vstupní funkce, výstupní funkce a vstupní inhibiční funkce*

P^{MS} *je množina všech multimnožin nad množinou P*

Pro naše účely budeme uvažovat speciální případ, kdy inhibiční funkce přiřazuje všem přechodům t∈T prázdnou multimnožinu z P^{MS} . V tomto případě není potřeba inhibiční funkci H ve struktuře PN uvádět.

Současný stav celého systému se znázorňuje pomocí značek zvaných tokeny, které se umisťují do jednotlivých míst. Systém vždy vychází z počátečního značení, což je výchozí značení před provedením jakéhokoliv přechodu. Formálně řečeno:

Definice 2.2 *Systém Petriho sítě (PN systém) je pětice* $\langle P, T, I, O, M0 \rangle$*, kde*

- $\langle P, T, I, O, \rangle$ *je struktura PN*
- *M0 je zobrazení typu* $P \rightarrow N$*, tzv. počáteční značení, kde N je množina přirozených čísel*

Obecně symbolem M označujeme jakékoliv zobrazení $P \rightarrow N$*.*

Přechod do nového místa je umožněn splněním podmínek přechodu, který určují počty odebíraných a umísťovaných tokenů. V grafu je to znázorněno ohodnocením orientovaných hran. Pokud není explicitně uvedeno hodnocení hrany, má implicitně hodnocení 1. Přechod se může uskutečnit tehdy, když všechna vstupní místa obsahují dostatečný počet tokenů.

Definice 2.3 *Přechod t je proveditelný při značení M, jestliže platí*

$$(\forall p \in {}^\bullet t)[M(p) \geq I(t,p)]$$

Kde

- ${}^\bullet t$ *je množina vstupních míst přechodu t*
- *M(p) je přirozené číslo určující počet tokenů v místě p*
- *I(t,p) je násobnost hrany z místa p do přechodu t*

Po provedení přechodu se tokeny odeberou ze vstupních míst v závislosti na ohodnocení orientovaných hran vystupujících ze vstupních míst a umístí do výstupních míst opět v závislosti na ohodnocení orientovaných hran, které do nich vstupují.

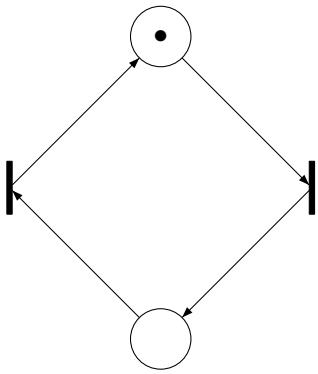


Figure 4: Jednoduchá síť po provedení přechodu

Množinu přechodů proveditelných při značení M označíme jako E(M). Značení, při kterém není žádný přechod proveditelný, označujeme *uzamčení*. Tomuto speciálnímu stavu systému se budeme věnovat později.

Definice 2.4 *Provedení přechodu t při značení M, vede od značení M k značení M' takovému, že platí:*

$$(\forall p \in P)[M'(p)=M(p)+O(t,p)-I(t,p)]$$

Kde O(t,p) je násobnost hrany z přechodu t do místa p

Provedení přechodu v sítích, které obsahují násobné hrany, je znázorněno na obrázku číslo 5.

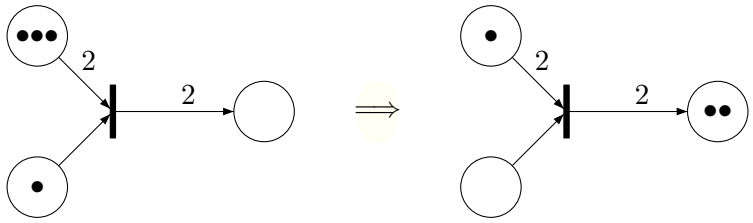


Figure 5: Provedení přechodu v síti s násobnými hranami

Jak je vidět na tomto obrázku, mohou se tokeny v systému Petriho sítí ztrácet. Může však nastat situace, kdy se tokeny mohou i generovat. Tyto vlastnosti využívají speciální struktury Petriho sítí, ke kterým se vrátíme později.

Vybaveni touto teorií se již můžeme vrátit k systému producent a konzument a ukázat si stav systému po provedení několika přechodů. Pro lepší názornost si všechna místa a přechody systému pojmenujeme. Systém již prošel několika cykly a nyní je podsystém producent ve stavu *připraven k dodání*, je plný sklad a podsystém konzument je připraven ke spotřebě. Popsanou situaci znázorňuje následující obrázek číslo 6.

Jak je vidět na obrázku číslo 6, je v uvedeném značení možné provést pouze jediný přechod a to *odběr*. Ostatní přechody není možné provést, protože některá jejich vstupní místa nemají dostatečný počet tokenů. Po provedení přechodu *odběr* se vyprázdní sklad a systém bude moci provést buď doplnění skladu, nebo akci spotřeby.

Reprezentace pomocí Petriho sítí umožňuje postihnout celý systém, provázání jednotlivých souběžných akcí, podmínky pro jejich provedení a zároveň pomocí tokenů můžeme nasimulovat i dynamiku celého systému a všechny jeho možné stavy. V neposlední řadě, díky grafové struktuře, můžeme použít celý aparát teorie grafů na analýzu tohoto systému a díky němu například určit, zda je systém *reverzibilní* (tedy je možné se

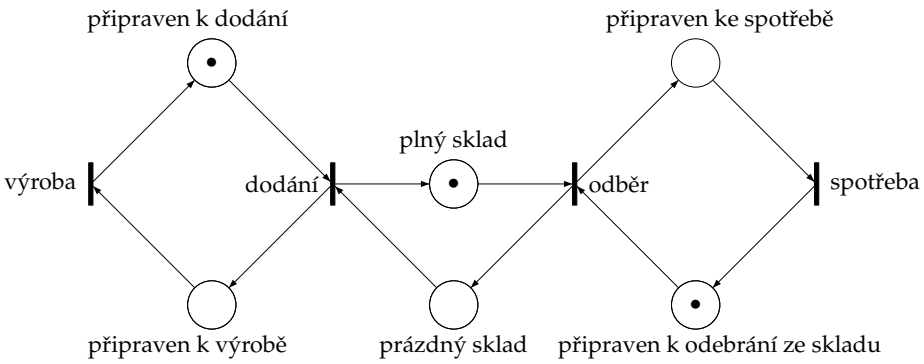


Figure 6: Producent - konzument po provedení přechodů

vrátit do počátečního stavu), nebo zda je systém *uzavřený* (jestliže z počátečního značení je dosažitelné značení, ve kterém není žádný přechod proveditelný.)

2.5 Vzájemné vyloučení

Dalším jednoduchým, ale často se vyskytujícím problémem v informatice je problém vzájemného vyloučení. Algoritmus vzájemného vyloučení, který byl poprvé popsán E. W. Dijkstrou v [5], zaručuje přístup ke sdílenému zdroji pouze jedinému procesu v daný okamžik. Tento problém se běžně vyskytuje v operačních systémech, databázích, počítačových sítích, všude tam kde je potřebné vyřešit konflikt, ke kterému dochází v případě pokusu o přístup k jedinému sdílenému zdroji více procesy.

Příkladem mohou být dva procesy, proces 1 a proces 2, které pracují nezávisle na sobě a každý z nich má určitou část, zvanou *kritická sekce*, do které musí mít volný přístup, ale vstoupit do ní může v jednom okamžiku pouze jeden z procesů.

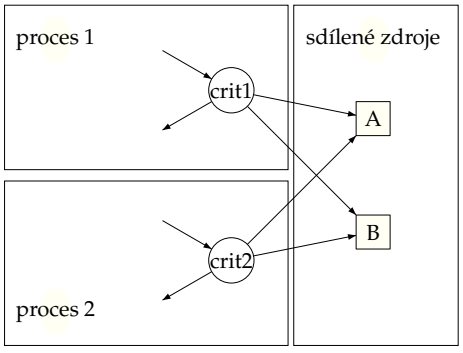


Figure 7: Vzájemné vyloučení

Takovou kritickou sekcí může být například přístup ke sdílené paměti, do které může přistupovat pouze jediný proces. Řešení tohoto problému zahrnuje nejen pouhé zajištění přístupu pouze jednoho procesu v daném okamžiku, ale je nutné vyřešit i další problémy s tímto spojené, ke kterým patří uváznutí v nějaké fázi průběhu algoritmu, nebo nemožnost faktického vstupu jednoho z procesů do své kritické sekce.

Nejjednodušším řešením je použití klíče, kdy pouze proces mající klíč může vstoupit do své kritické sekce. Pro formální zápis programovým pseudokódem budeme uvažovat dva procesy P1 a P2 a proměnnou klíč. Zápis vypadá takto:

```
(klíč : integer; klíč := 1);
begin
   parbegin
      proces 1:   begin P1:   if klíč := 0 then goto P1;
                        klíč := 0
                        kritická sekce 1
                        klíč := 1
                        zbytek procesu 1; goto P1
                  end;
      proces 2:   begin P2:   if klíč := 0 then goto P2;
                        klíč := 0
                        kritická sekce 2
                        klíč := 1
                        zbytek procesu 2; goto P2
                  end;
   parend
end;
```

Výpis 2: Vzájemné vyloučení

Již z tohoto zápisu je zřejmé, že zde chybí jakýkoliv mechanismus, řídící přidělování klíče procesům. Pokud se podíváme na grafickou reprezentaci tohoto distribuovaného algoritmu pomocí Petriho sítě, dojdeme ke stejnému zjištění. Pokud proces 1 vezme klíč a vstoupí do své kritické sekce, nemůže proces 2 udělat to samé.

Avšak toto triviální řešení s sebou nese několik problémů, zejména chybějící mechanismus přidělování klíče a také možnost výskytu situace, že jeden z procesů se do své sekce nemusí nikdy dostat.

Algoritmu vzájemného vyloučení se budu věnovat podrobněji v kapitole číslo 3, ale tato jednoduchá síť v sobě skrývá zajímavou strukturu, kterou si blíže popíšeme již nyní.

V počátečním stavu jsou oba vstupy do kritických sekcí proveditelné, protože je ve vstupních místech dostatečný počet tokenů pro provedení přechodu. Jak již bylo řečeno, pokud ale jeden z procesů převezme klíč a vstoupí do kritické sekce, odstraní tímto token z místa **klíč** a zamezí tak druhému procesu v provedení vlastního přechodu do své kritické sekce. Formálně se tato situace nazývá *konflikt*.

Je to stav, kdy provedení jednoho přechodu snižuje stupeň proveditelnosti druhého přechodu. Konflikty rozdělujeme na *symetrické* (provedení kteréhokoliv přechodu sníží stupeň přechodu toho druhého) a *asymetrické* (provedení jednoho přechodu sníží stupeň

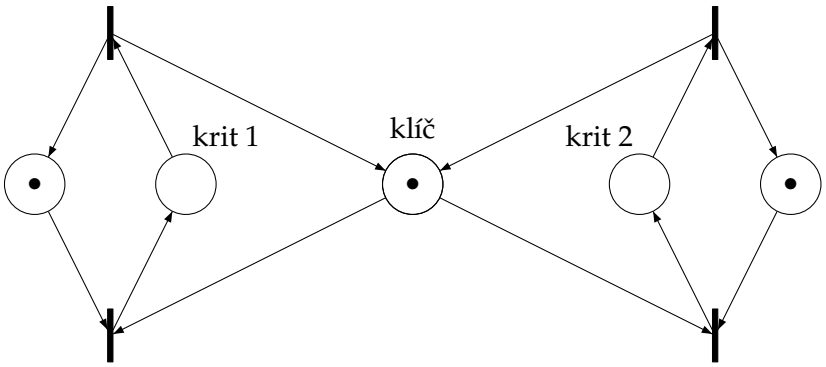


Figure 8: Algoritmus vzájemného vyloučení

proveditelnosti druhého přechodu, ale ne naopak). V našem případě se tedy jedná o konflikt symetrický.

2.6 Sdílení zdrojů

Zobecněním problému vzájemného vyloučení dostáváme problém vzájemného sdílení více zdrojů. Příkladem může být proces vyžadující pro své spuštění více prostředků, jako je tiskárna spolu s databází a spolu se síťovým portem.

Všeobecně známou a populární verzí tohoto problému je konfigurace několika subsystémů, kde je každý zdroj sdílen dvěma subsystémy a zároveň každý z těchto subsystémů vyžaduje dva zdroje. Tento speciální případ byl popsán poprvé E. W. Dijsktrou [5] jako problém hladovějících filozofů. V tomto problému filozofové představují jednotlivé procesy a jídelní hůlky představují jejich zdroje. Filozofové sedí u společného stolu a po obou jejich stranách leží hůlky potřebné k jídlu. Z tohoto nastavení vyplývá, že žádný ze sousedících filozofů nemůže jíst zároveň. Úkolem tohoto algoritmu je zajistit, aby každý z filozofů měl spravedlivý přístup k jídelním hůlkám, tedy aby každý proces měl možnost se dostat ke sdíleným zdrojům.

Jednoduchý algoritmus popisující život filozofa vypadá takto:

```
cycle begin přemýšlej;
            uchop (pravou hůlku);uchop (levou hůlku);
            jez;
            polož (pravou hůlku);polož (levou hůlku);
end;
```

Výpis 3: Život filozofa

Toto jednoduché řešení však opět obsahuje problém spravedlivého přístupu ke zdrojům a navíc může dojít k *uváznutí* (uzamčení) systému. Tento speciální stav je možné demonstrovat na Petriho síti tohoto algoritmu. Z důvodu zjednodušení znázorňuje následující Petriho síť pouze čtyři procesy.

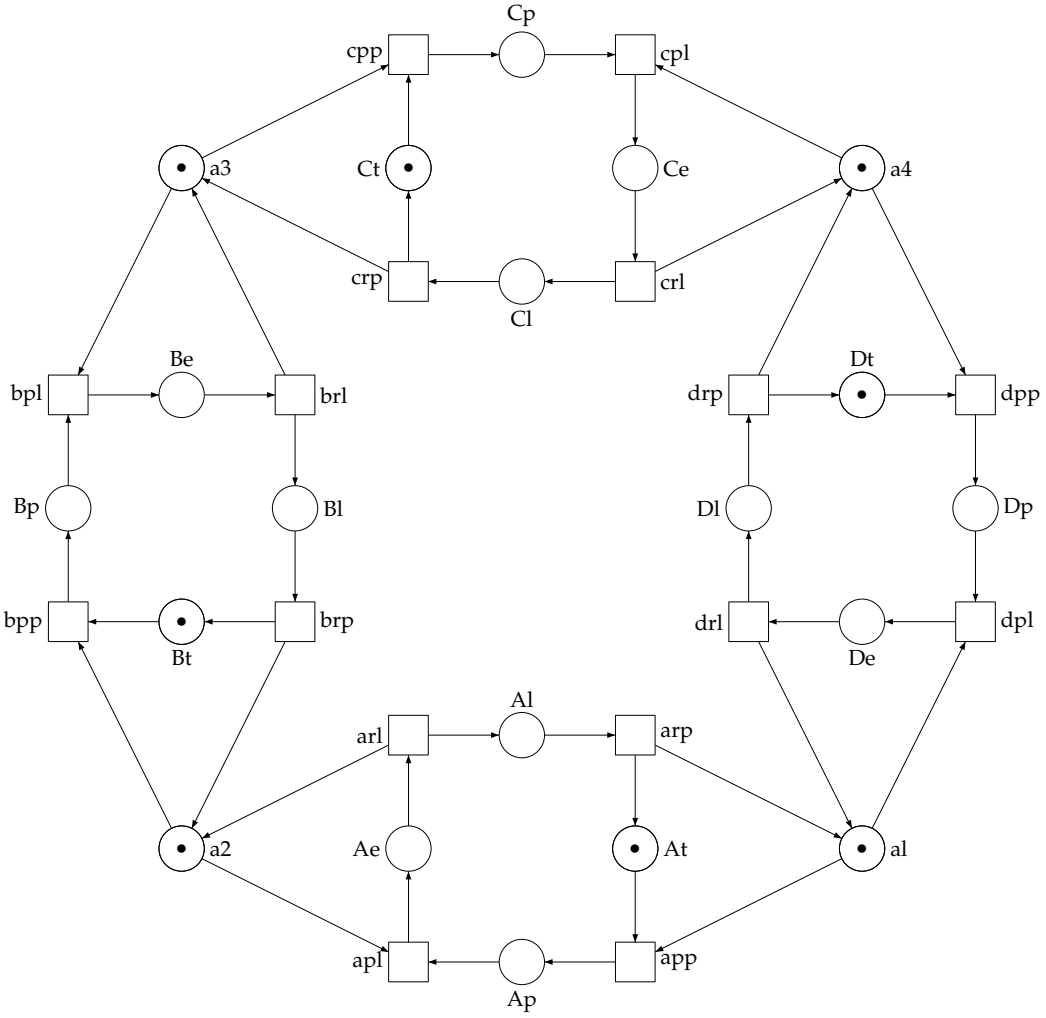


Figure 9: Distribuovaný algoritmus hladových filozofů

Pro lepší pochopení si tento systém popíšeme. Filozofy si označíme **A**, **B**, **C**, **D** a ti mohou být v jednom ze čtyř stavů. Například filozof A prochází stavy **At** - přemýšlí (počáteční stav každého filozofa. Je ohodnocen jedním tokenem), **Ap** - uchopil pravou hůlku, **Ae** - jí (po uchopení druhé hůlky), **Al** - odevzdal levou hůlku. Dále zde máme reprezentovány samotné jídelní hůlky - **a1**, **a2**, **a3**, **a4**. Token v těchto místech značí možnost uchopení filozofem.

Uvedená Petriho síť obsahuje symetrické konflikty právě při procesech uchopení jednotlivých hůlek. Proces, který ji uchopí, znemožní svému sousedovi vstup do stavu *jí*. Protože zde není žádný mechanismus řízení převzetí hůlek, může nastat situace, kdy pouze proces A a proces C budou nekonečně dlouho probíhat a na proces B a D nikdy nepřijde řada.

Druhé nebezpečí, které tato konstrukce skrývá, se nazývá *uváznutí* (deadlock). Obecně to je situace, kdy žádný z procesů nemůže pokračovat ve své práci. V některých případech je uzamčení žádoucí, tedy vyžadujeme, aby například program po určitém počtu kroků skončil. Naopak u operačního systému vyžadujeme, aby běžel nepřetržitě, tedy neobsahoval žádná uzamčení.

Vrátíme-li se k našemu příkladu, jistě vyžadujeme, aby měl každý proces možnost využít sdílené prostředky a navíc s relativně spravedlivou četností. V tomto algoritmu dojde k uváznutí v okamžiku, kdy každý z filozofů uchopí svou pravou hůlku, tedy každý z procesů přejde do stavu *p*. V tuto chvíli jsou procesy uvázlé, protože pro přechod do další fáze potřebují přítomnost své levé hůlky, ale ta již není volná. Tuto situaci znázorňuje následující fragment Petriho sítě.

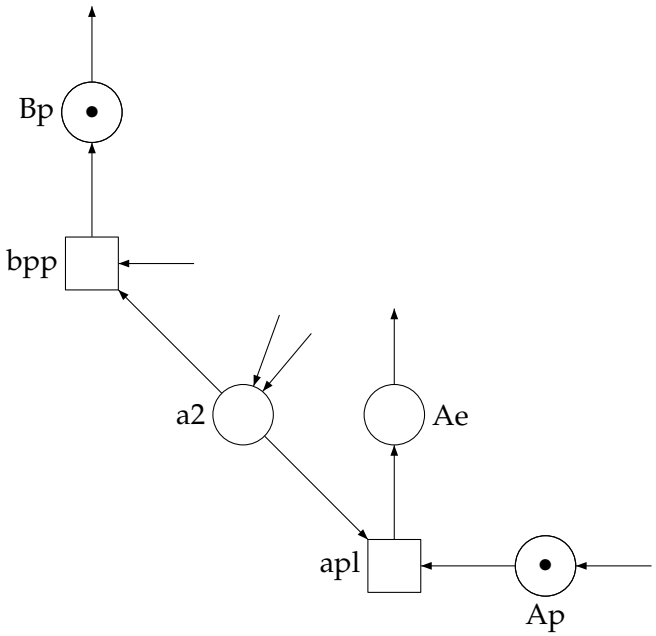


Figure 10: Uváznutí v algoritmu hladových filozofů

Filozof A i B uchopil svou pravou hůlku, tedy i filozof B uchopil hůlku a2. Nyní budou procesy pouze nekonečně dlouho uváznuté v situaci, kdy druhou hůlku nemohou dostat.

2.7 Stavová analýza Petriho sítí

Abychom si formálně popsali situaci uváznutého systému, popíšeme si několik vlastností Petriho sítí potřebných k jejich stavové analýze.

- Značení M' je *dosažitelné* ze značení M, jestliže existuje posloupnost přechodů, která je proveditelná ve značení M a která převádí Petriho síť ze značení M do značení M'.
- Značení M Petriho sítě se nazývá *vždy dosažitelným*, jestliže je dosažitelné z každého dosažitelného značení.
- PN systém se nazývá *reversibilní*, je-li počáteční značení vždy dosažitelné, a tedy i každé libovolné dosažitelné značení je dosažitelné z libovolného dosažitelného značení.
- PN systém se nazývá systémem **bez uzamčení**, jestliže z počátečního značení M0 není dosažitelné žádné značení, ve kterém není žádný přechod proveditelný.
- PN-systém je *živý*, jestliže při vývoji systému žádný přechod nikdy neztrácí možnost, že bude někdy v budoucnu znovu proveden, tedy z každého dosažitelného značení je možné spustit výpočetní posloupnost obsahující všechny přechody.
- Místo p PN-systému se nazývá *k-omezené* (k-bounded), jestliže pro každé dosažitelné značení je počet tokenů v tomto místě nanejvýše rovný k.
- PN-systém se nazývá *k-omezený*, jestliže všechna jeho místa jsou k-omezená.
- PN-systémy se nazývají *bezpečnými* (safe), jestliže jsou 1-omezené.

Metoda stavové analýzy PN systému $\langle P, T, I, O, M0 \rangle$ se zakládá na konstrukci množiny dosažitelných značení RS(M0)a grafu dosažitelnosti RG.

Definice 2.5 *Množina dosažitelnosti PN systému* $\langle P, T, I, O, M0 \rangle$ *je množina RS(M0) definovaná induktivně takto:*

- *M0*$\in$*RS(M0)*
- *M0*$\in$*RS(M0)* $\bigwedge$ $(\exists t \in T) M \xrightarrow{t} M'$

Kde $M \xrightarrow{t} M'$ *značí že značení M' je bezprostředně dosažitelné ze značení M.*

Graf dosažitelnosti PN systému s množinou dosažitelnosti RS(M0) je hranově ohodnocený orientovaný graf RG(M0) s následujícími vlastnostmi:

- *RG(M0) je množina uzlů grafu*
- *Množina hran A grafu je definována takto:*

- $A \subseteq RS\ x\ RS\ x\ T$
- $\langle M_i, M_j, t \rangle \in A \iff M_i \underset{\rightarrow}{t} M_j$

- *M0 je počáteční uzel grafu*

Máme-li popsán PN systém grafem dosažitelnosti, redukuje se problém analýzy PN-systému na problém analýzy orientovaného grafu. Platí následující vlastnosti:

- Značení M' je dosažitelné ze značení M $\iff$ V grafu RG vede orientovaná cesta z uzlu M do uzlu M'
- Značení M je v PN systému vždy dosažitelným stavem $\iff$ Z každého uzlu grafu RG vede orientovaná cesta do uzlu M
- PN systém je reversibilní $\iff$ V grafu RG vede z každého uzlu orientovaná cesta do uzlu M0 $\iff$ V RG grafu vede orientovaná cesta z každého uzlu do každého - graf je silně souvislý
- PN systém neobsahuje uzamčení $\iff$ V RG grafu neexistuje uzel, ze kterého by nevedla žádná hrana
- PN systém je živý $\iff$ Pro všechny koncové silně souvislé komponenty RG grafu platí: každý přechod ohodnocuje aspoň jednu hranu komponenty
- PN systém je omezený $\iff$ Graf RG je konečný

Podrobnější prohlídka grafu dosažitelnosti umožňuje také identifikovat vzájemnou výlučnost míst a přechodů, nebo nám umožní nalézt takové značení, při kterém vznikají konflikty.

2.8 Sdílení zdrojů - pokračování

Podíváme-li se nyní na Petriho síť algoritmu hladových filozofů z obrázku číslo 9, můžeme o něm říct, jakých nabývá vlastností:

- Všechna místa systému mohou získat pouze 1 token. $\Longrightarrow$ Systém je tedy 1-omezený a bezpečný
- Systém obsahuje značení, ve kterém není proveditelný ani jeden přechod. $\Longrightarrow$ Systém tedy obsahuje uzamčení.
- Při vývoji systému může dojít k okamžiku (uzamčení) kdy všechny přechody ztrácí možnost v budoucnosti. $\Longrightarrow$ Systém není živý.
- Díky uzamčení systému není možnost návratu do počátečního značení. $\Longrightarrow$ Systém tedy není reversibilní.

Jak z této analýzy vyplývá, není takto navržený systém optimální. Pokusíme se tedy odstranit uváznutí, které je závažnějším problémem než nespravedlivé rozdělování sdílených zdrojů. Z uvedeného algoritmu vyplývá, že k uváznutí dojde ve chvíli, kdy každý proces symetricky zabere jednu ze sdílených hodnot. Řešením proto musí být systém, který sdílené zdroje zabírá v jediném okamžiku. K mechanismu kontroly možnosti zabrání obou sdílených zdrojů se vrátíme později. Petriho síť upraveného distribuovaného algoritmu potom vypadá takto:

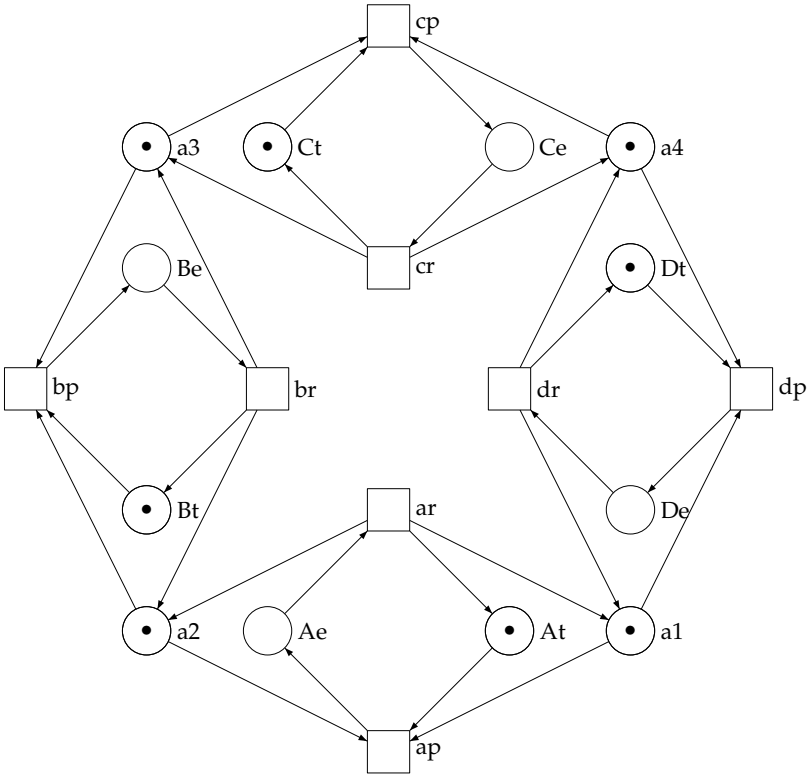


Figure 11: Algoritmus hladových filozofů bez uváznutí

Z Petriho sítě je patrné, že algoritmus nemá uváznutí, protože pokud chce přejít do stavu e, musí mít možnost vzít oba sdílené zdroje. Nicméně stále nemáme zajištěný spravedlivý přístup všech procesů ke sdíleným zdrojům.

Po provedení stavové analýzy docházíme ke zjištění, že uvedený systém má změněné některé vlastnosti:

- Všechna místa systému mohou stále získat pouze 1 token. $\Longrightarrow$ Systém je tedy 1-omezený a bezpečný

- Systém již neobsahuje značení, ve kterém není proveditelný ani jeden přechod. $\Longrightarrow$ Systém tedy neobsahuje uzamčení.
- Při vývoji tohoto systému nemůže nastat situace, kdy nějaký přechod ztrácí možnost budoucího provedení. $\Longrightarrow$ Systém je tedy živý.
- V systému vždy existuje sekvence přechodů, která vrací systém do počátečního značení $\Longrightarrow$ Systém je tedy reversibilní.

Nyní se vrátíme k mechanismu, který nám zaručí, že při pokusu o získání obou sdílených proměnných nedojde k uváznutí procesů. Jedním z možných řešení je Lehmann-Rabinův algoritmus pojmenovaný po jeho tvůrcích [7]. Každý proces zde pracuje pouze se sousedícími zdroji, na které se odvolává pomocí relativních jmen - *Res(i,zleva)* a *Res(i,zprava)* s hodnotami volná, obsazená. Dále definujeme lokální proměnou $u_i \in$ zleva, zprava a operátor *opp*, který je doplňkem k hodnotě svého argumentu, t.j. *opp*(zprava) = zleva a *opp*(zleva) = zprava

```
cycle begin přemýšlej;
            T:   ui := random (náhodně vybráno zleva či zprava)
            P1:  if Res(i,ui) = volná then Res(i,ui) := obsazená;
                 else goto P1;
            P2:  if Res(i,opp(ui)) = volná then Res(i,opp(ui)) := obsazená; goto E;
            E:   jez i
                 ui := random (náhodně vybráno zleva či zprava)
                 Res(i,ui) = volná; Res(i,opp(ui)) = volná;
end;
```

Výpis 4: Lehmann-Rabinův algoritmus

Část tohoto fragmentu pseudokódu si můžeme pro lepší porozumění zobrazit také jako Petriho síť pro proces A.

Tento algoritmus již obsahuje mechanismus, který v případě uchopení jedné hůlky a zjištění, že druhá je obsazená, vrátí svoji hůlku zpět do volné a začne s novou kontrolou přístupu ke sdíleným hůlkám. V tomto případě nedojde k úplnému uzamčení, protože procesy jsou případně restartovány. Obrázek 12 znázorňuje pouze proces uchopení hůlek, proces jejich vrácení je triviální.

Pokud bychom chtěli tyto detailní struktury začlenit do výše zmíněné struktury algoritmu hladových filozofů z obrázku 11, zjistili bychom, že výsledná struktura je značně nepřehledná. Toto je také jeden z omezujících faktorů využití Petriho sítí pro modelování složitých distribuovaných systémů.

Tento nedostatek můžeme částečně kompenzovat použitím metody transformace grafu. Pro lepší pochopení tohoto problému si tuto transformaci nejdříve formálně definujeme.

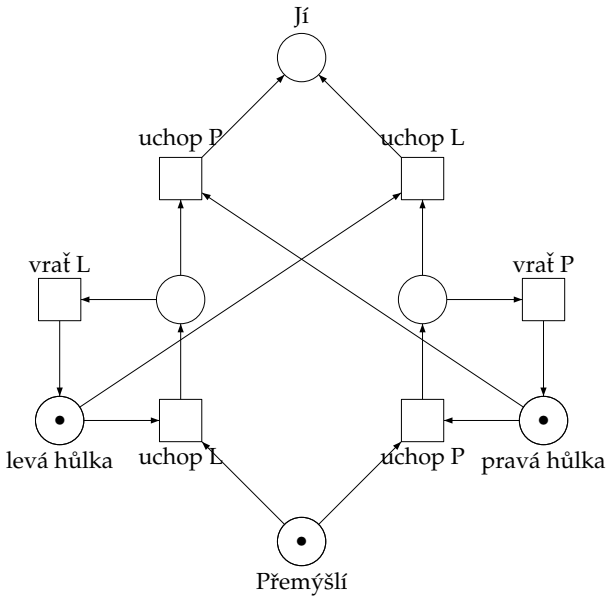


Figure 12: Fragment Lehmann-Rabinova algoritmu uchopení hůlek

2.9 Grafová transformace Petriho sítě

Pro popsání grafové transformace Petriho sítě využijeme modely algoritmů z předešlých kapitol. Využijeme toho, že je Petriho síť orientovaný bipartitní graf a na jeho uzly použijeme metodu redukce. Využíváme redukční pravidla pro transformaci systému na jednodušší, tedy na systém, jehož analýza bude časově méně náročná, případně triviální. Použitá redukční pravidla musí být vhodně zvolena tak, aby zachovávala vybrané vlastnosti PN systémů. Příkladem může být živost, reverzibilita nebo omezenost. Systém vzniklý pomocí takovýchto redukčních pravidel pak bude mít stejné vlastnosti jako systém originální. Následující obrázek představuje pravidla spojení sekvence míst a spojení sekvence přechodů.

K pravidlu spojení sekvence míst je nutné dodat, že každá případná další vstupní hrana místa p1 je také vstupní hranou místa p12 a podobně každá výstupní hrana místa p2 je i výstupní hranou místa p12. Počet tokenů v místě p12 musí být zároveň roven součtu tokenů v místech p1 a p2. V případě spojení sekvence přechodů platí stejná poznámka ohledně vstupních a výstupních hran, a navíc místo mezi oběma přechody nesmí obsahovat žádné tokeny.

Další možnou metodou transformace složitějšího systému na systém jednodušší je spojení podobných struktur. V tomto případě se podobné části struktury Petriho sítě spojí v jednu, která je na vyšší úrovni. Její části jsou pak reprezentovány popisem jednotlivých uzlů a hran. Tato metoda je popsána na následujících obrázcích číslo 14 a 15, které popisují diagram systému producent - konzument pro více objektů.

V případě, že bychom chtěli mít diagram pro 10 objektů, nebylo by již příliš vhodné postupovat takto. Proto spojíme podobné struktury do struktury jediné. Výsledkem je

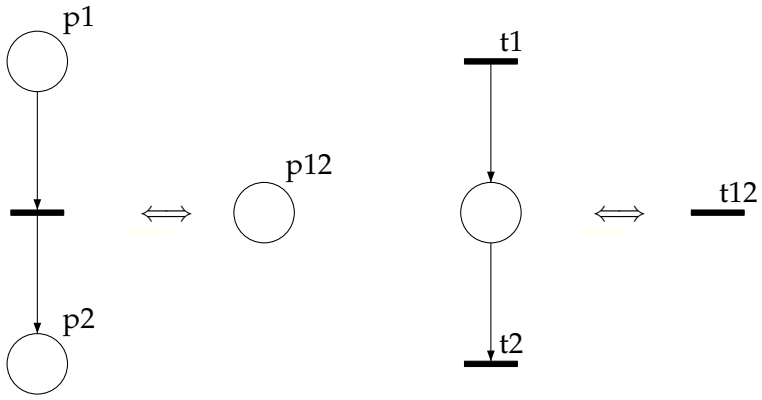


Figure 13: Spojení sekvence míst a přechodů

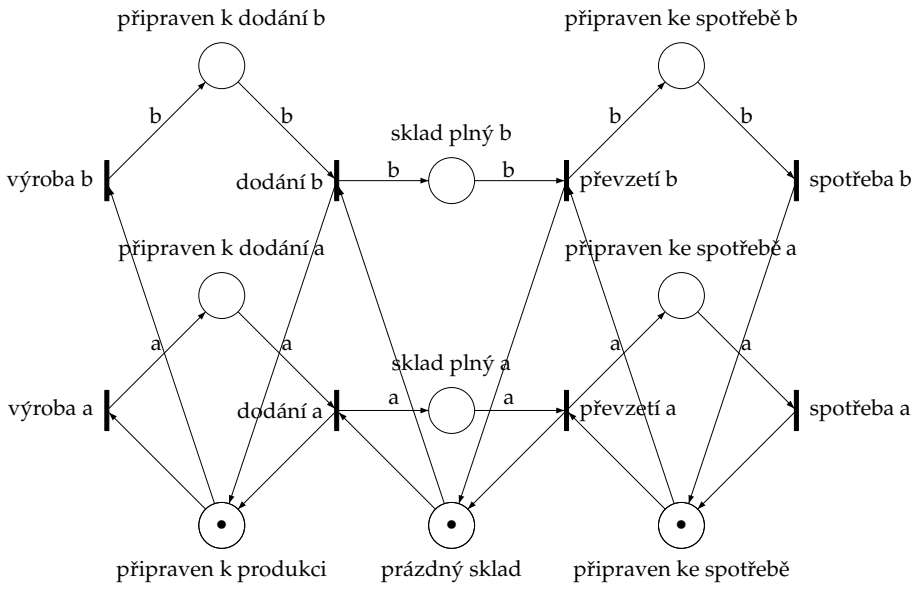


Figure 14: Systém producent - konzument pro dva objekty

diagram systému producent - konzument pro jakýkoliv objekt, který je znázorněn na obrázku číslo 15.

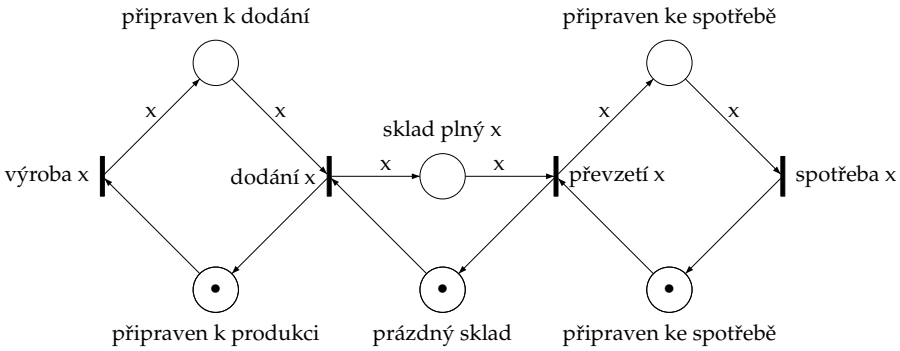


Figure 15: Systém producent - konzument pro x objektů

Nyní se můžeme vrátit k algoritmu hladových filozofů. Pro snazší pochopení principu transformace si tento diagram upravíme pouze pro tři filozofy A,B,C a troje hůlky f1,f2,f3, tak jak je zobrazen na obrázku číslo 16.

V tomto diagramu můžeme nyní identifikovat tři "podobné" skupiny míst a dvě "podobné" skupiny přechodů. Na základě výše uvedené metody spojení podobných míst můžeme vytvořit nový systém, který má již pouze tři místa nazvaná *obědvající filozofové*, *přemýšlející filozofové* a *volné hůlky*. Grafické znázornění tohoto systému ukazuje obrázek číslo 17.

Poslední věc, která nám zbývá pro zobecnění algoritmu, je spojit i podobná místa. Pro tento účel si take definujeme funkce, které jednotlivým filozofům přidělují levou a pravou hůlku:

- l(a) = r(b) = f1
- l(b) = r(c) = f2
- l(c) = r(a) = f3

Tyto funkce můžeme zobecnit na l(x) a r(x). Výsledný algoritmus je znázorněn na obrázku číslo 18.

Vidíme, že tento diagram je mnohem jednodušší, s větší mírou abstrakce a zobecnění. Pro následující algoritmy se již omezíme, z důvodu vyšší složitosti struktury, pouze na tyto typy diagramů.

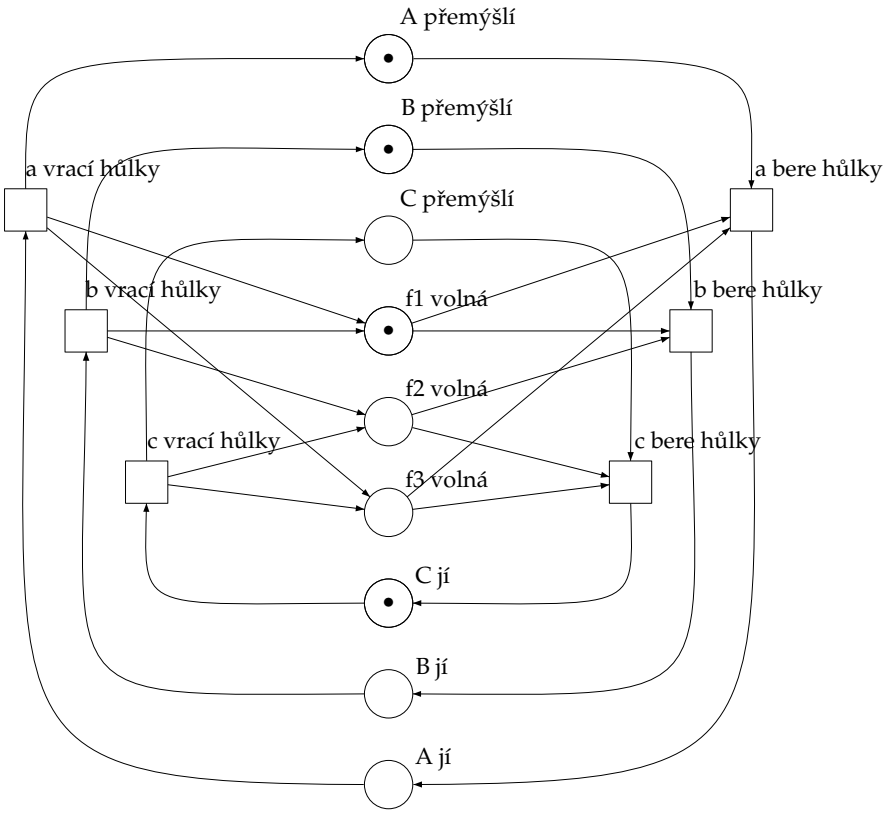


Figure 16: Algoritmus hladových filozofů pro 3 filozofy

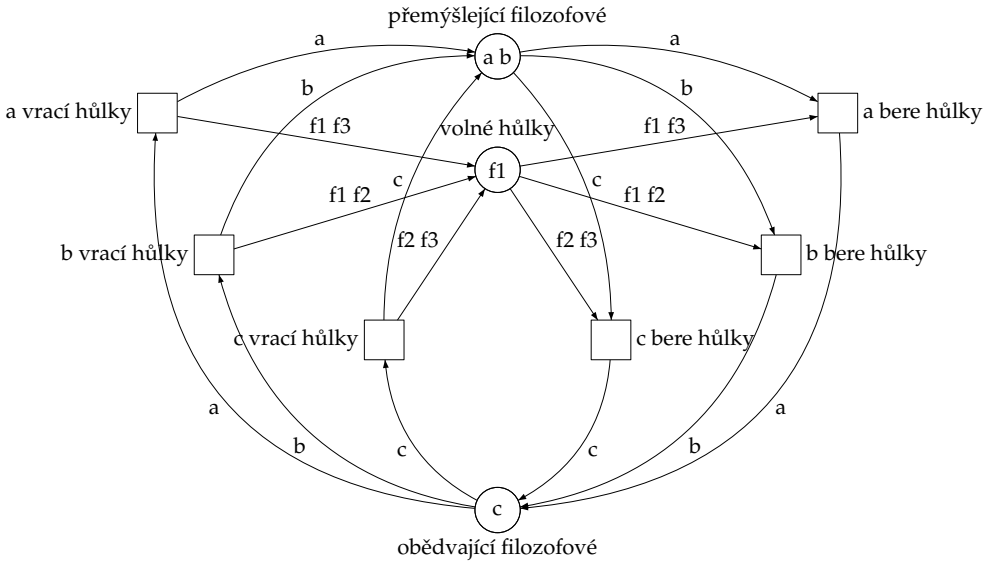


Figure 17: Algoritmus hladových filozofů se spojenými místy

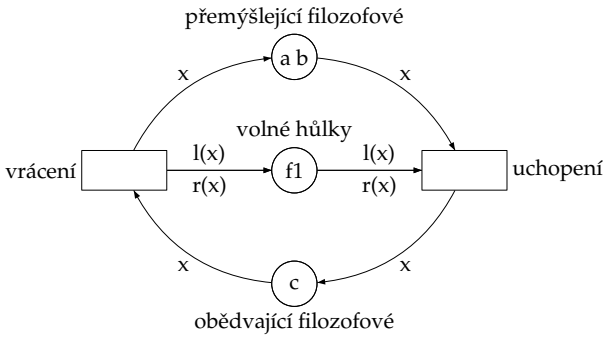


Figure 18: Zobecněný algoritmus 3 hladových filozofů

2.10 Výběr vedoucího

Tento známý a používaný algoritmus se poprvé objevuje v lokálních sítích typu *token ring*. Tyto kruhové sítě fungují na principu předávání speciálního rámce (tzv. *tokenu*) mezi uzly. Uzel, vlastnící tento token, má jako jediný právo vysílat. Může se ale stát, že je tento token ztracen, a potom nastává potřeba nějakým způsobem tento token znovu vytvořit.

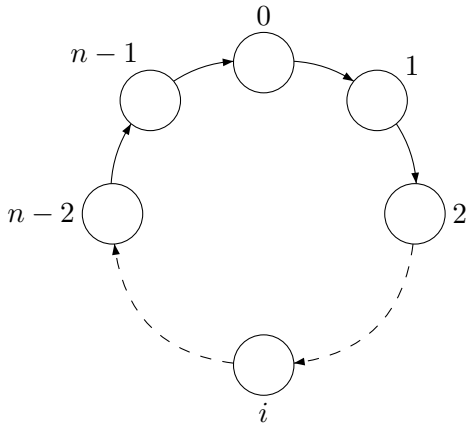


Figure 19: Síť typu token ring

Pokud se na tyto uzly podíváme jako na procesy pracující v jedné síti, můžeme tento princip rozšířit i mimo pouhé vysílání zpráv na problém řízení práce všech těchto uzlů v rámci distribuované sítě.

Jednoduchým řešením tohoto problému je takzvaný *"Bully"* algoritmus popsaný v [8]. Tento algoritmus je možné použít pouze v případě očíslovaných procesů. Algoritmus začíná v okamžiku, kdy jakýkoliv proces P zjistí, že po pevně určené době nedostal

zprávu od aktuálního vedoucího, nebo s ním nemůže navázat spojení. Od této chvíle začíná sekvence kroků tohoto algoritmu.

- P rozešle zprávu o volbě všem procesům, které mají větší ID.
- pokud P nedostane zpět zprávu od žádného procesu s větším ID, vyhrává volbu a rozešle zprávu o vítězství.
- pokud P dostane zpět zprávu od nějakého procesu s větším ID, vyčká pevně určenou dobu, zda se proces s vyšším ID prohlásí za vítěze. Pokud tuto zprávu nedostane, spustí nové kolo volby.

V případě, že proces dostane zprávu o volbě od procesu s nižším ID, ihned spustí nové kolo volby.

Dalším známým řešením tohoto problému je Chang-Roberts algoritmus, poprvé popsaný v [9]. Tento algoritmus předpokládá, že každý proces má univerzální ID (UID) a jednotlivé procesy mohou vytvořit kruhovou síť. Algoritmus se skládá ze dvou částí.

- Každý proces je označen jako *nezúčastněný*
- Uzel, který zjistí nepřítomnost vedoucího, zahájí volbu. Označí se za *zúčastněného* a odešle *zprávu o zahájení volby* po směru hodinových ručiček se svým UID.
- Když proces obdrží *zprávu o zahájení volby*, porovná své UID s UID ve zprávě volby. Pokud má své aktuální UID větší než to ve zprávě, přepíše UID ve *zprávě o zahájení volby* svým a odešle ji po směru hodinových ručiček. Označí se za *zúčastněného*.
- V případě že tuto *zprávu o zahájení volby* obdrží již za stavu *zúčastněný*, je postup odlišný. Opět porovná své UID s UID ve zprávě, ale odešle tuto zprávu dále pouze v případě, že je nutné UID zprávy přepsat.

První část algoritmu skončí ve chvíli, kdy proces obdrží *zprávu o zahájení volby* se svým UID. V tuto chvíli se spouští část druhá.

- Tento vítězný proces se označí jako *nezúčastněný* a odešle *zprávu o výsledku volby*, oznamující jeho zvolení a jeho UID svému sousedovi.
- Když uzel obdrží zprávu o výsledku volby, označí se jako *nezúčastněný*, zaznamená si UID zvoleného vedoucího a *zprávu o výsledku volby* pošle zase dál.
- Když zpráva o výsledku volby doputuje k nově zvolenému vedoucímu, je tento algoritmus výběru vedoucího ukončen.

Variantou těchto řešení je algoritmus, který z celkové množiny možných kandidátů jejich postupným porovnáváním odstraňuje nevhodné kandidáty. V závěru výpočtu tak zůstává pouze jediný, nejlepší kandidát. Princip tohoto algoritmu reprezentuje Petriho síť na obrázku číslo 20 uvedená v [1].

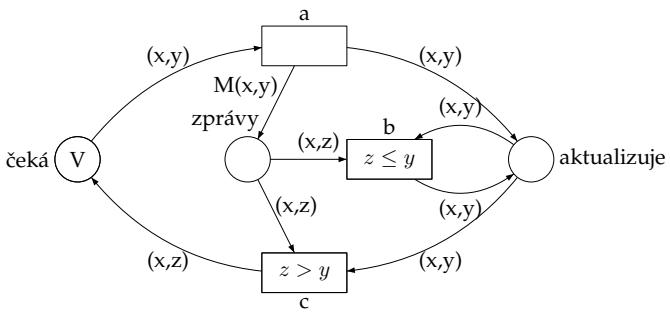


Figure 20: Výběr vedoucího

Na začátku jsou všechny místa ve stavu *čeká*. Později se tu budou objevovat stále lepší kandidáti na vedoucího. Jakékoliv místo x z čekajících může pomocí akce **a** dát vědět svým sousedům o sobě a svém aktuálním kandidátu na vedoucího y. Tím se místo stane *aktualizující*. Aktualizující místo *(x,y)* může dostat zprávu o novém kandidátu *(x,z)*. Pokud tento nově navržený kandidát nepřesáhne stávajícího kandidáta, místo *(x,y)* zůstane aktualizující pomocí akce *b*. V opačném případě se do *čekajících* míst vrátí s novým kandidátem z pomocí akce *c*. Na konci pak zůstanou pouze dvojice složené ze všech míst a nejlepšího kandidáta.

2.11 Minimální kostra grafu

Posledním algoritmem, kterému se budu věnovat v této kapitole, je algoritmus hledání minimální kostry grafu. Podgraf grafu, který spojuje všechny uzly a je stromem, se nazývá *kostra grafu*. Graf může mít několik rozdílných koster. Pokud jsou hrany grafu ohodnoceny, součet těchto ohodnocení se nazývá *váha grafu*. V takovémto grafu s ohodnocenými hranami je možné najít minimální kostru grafu, která má váhu menší nebo rovnu vahám všech ostatních koster tohoto grafu.

Hledání minimální kostry grafu se používá všude tam, kde je vhodné minimalizovat zdroje potřebné na komunikaci mezi procesy v rámci sítě. Další využití tohoto algoritmu představují aplikace s komplexními sítěmi a množstvím potencionálních kontrolních problémů. Využitím vysílání pouze po kostře těchto sítí je výrazně snížena jejich komplexnost a tím pádem i snížený výskyt problémů.

Distribuované řešení tohoto problému spočívá v komunikaci mezi jednotlivými uzly [10]. Na počátku jednotlivé uzly znají váhu pouze přilehlých hran. Každý z uzlů pak provádí lokální algoritmus, který se sestává z posílání zpráv po těchto hranách a jejich zpracování. Poté je schopen určit, které hrany tvoří minimální kostru grafu.

Uvádím zde tento algoritmus pro demonstraci vhodnosti použití grafové reprezentace, jak ji uvádí W. Reisig ve své monografii [1]. Využívá k tomuto účelu výše popsaný algoritmus Výběru vedoucího. Tento algoritmus je ukončen s tím, že všechny uzly znají svého vedoucího. Pokud se k této informaci připojí vzdálenost k němu a bližší soused

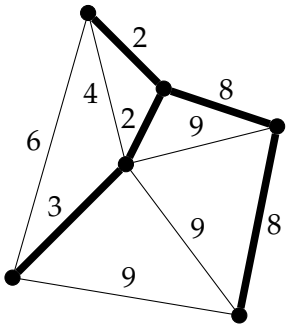


Figure 21: Minimální kostra

vzhledem k vedoucímu, může uzel komunikovat s vedoucím právě přes tohoto souseda. Linky spojující takto uzly s vedoucím přes bližší sousedy tvoří minimální kostru grafu. Tento algoritmus reprezentuje diagram z obrázku číslo 22.

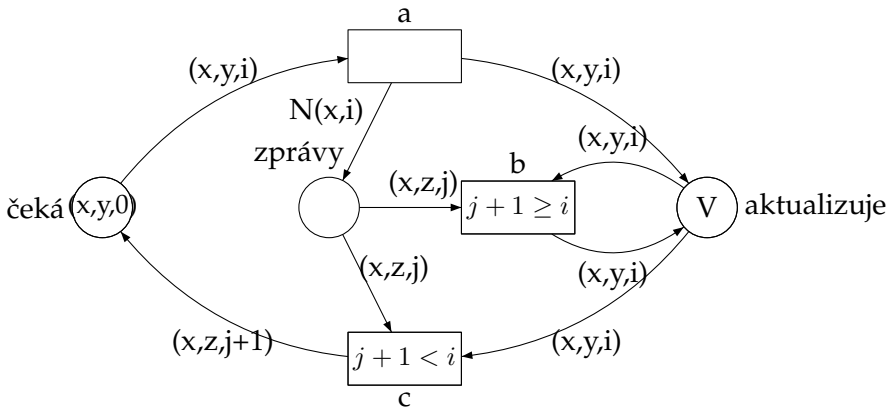


Figure 22: Minimální kostra grafu

Na počátku vedoucí r *čeká* s délkou 0 k vedoucímu. Ostatní uzly jsou v místě *aktualizace*, zatím bez kandidáta na vedoucího a s nekonečnou vzdáleností. V pozdějších fází obsahuje místo *čeká* token (x,y,i), kde je reprezentována trasa délky *i* z *x* přes *y* k vedoucímu. *Čekající* uzel x zašle svou aktuální vzdálenost i všem svým sousedům pomocí akce **a** a stane se *aktualizujícím*. Aktualizující uzel může obdržet zprávu (x,z,j). Pokud vzdálenost ve zprávě *j* od *z* k vedoucímu nevylepší aktuální vzdálenost *i*, zůstává uzel *x*

u vzdálenosti *i* přes *y* pomocí akce *b*. V opačném případě změní *x* vzdálenost na *j+1* přes *z* pomocí akce *c*.

Tento algoritmus potvrzuje vhodnost použití Petriho sítí pro reprezentaci distribuovaných algoritmů, která přináší na první pohled podobné struktury, které by nebyly tak zřejmé při použití jiných formalismů.

3 Vzájemné vyloučení - model

3.1 Úvod

V této kapitole si vytvoříme důkladný model algoritmu vzájemného vyloučení podrobně popsaný v [6].V každém kroku si popíšeme situaci, ukážeme si návrh řešení pomocí zápisu v pseudokódu a pro srovnání i pomocí Petriho sítí, které je pro modelování použito i v monografii Reisig, W.: Elements of distributed algoritms (Springer 1998). Pro zápis v pseudokódu použiji následující příkazy:

- begin je začátek procesu
- parbegin je začátek paralelních procesů
- parend je konec paralelních procesu
- end je konec procesu

Příklad: begin S1, parbegin S2, S3, S4, parend, end

3.2 Problém

Synchronizace je v informatice fundamentální výzvou. Stává se hlavní otazkou výkonu a návrhu souběžného programování v moderních architekturách.

Souběžný přístup ke sdíleným prostředkům několika procesy musí být řízen, aby nedošlo k nežádoucímu rušení mezi konfliktními operacemi. Algoritmy vzájemného vyloučení jsou mechanismy sloužící právě pro řízení těchto souběžných přístupů. Proces může přistoupit ke sdílenému zdroji pouze uvnitř kritické sekce, v níž má také garantován exkluzivní přístup. Tento přístup je značně populární zejména díky jednoduchosti svého modelu a možnostem implementace, které jsou efektivní a přenositelné. Všechny současné souběžné programy využívají nějaký ze systémů vzájemného vyloučení pro svou synchronizaci. Tento problém se běžně vyskytuje v operačních systémech, databázích, počítačových sítích a všude tam, kde je potřebné vyřešit konflikt, ke kterému dochází v případě pokusu o přístup k jedinému sdílenému zdroji více procesy.

Formálně je tento proces definován takto: každý z procesů vykonává sadu instrukcí v nekonečné smyčce. Tyto instrukce jsou seskupeny do čtyř skupin: zbytek, vstup, kritické sekce, výstup.

Programový zápis této struktury vypadá následovně:

```
loop forever;
    kód zbytku;
    kód vstupu;
     kritická  sekce};
    kód výstupu;
end loop
```

Výpis 5: Popis problému vzájemného vyloučení

Proces vždy začíná spuštěním zbytkového kódu. V určitý okamžik může nastat situace, ve které potřebuje spustit kód v kritické sekci. Pro vstup do kritické sekce musí proces projít celou procedurou, která zajistí, že během zpracovávání své kritické sekce žádný jíný proces nespustí svoji kritickou sekci. Po výstupu z kritické sekce spustí výstupní kód, který ostatním procesům oznámí, že již déle není ve své kritické sekci. Po té se proces vrátí ke zbytku kódu.

Problém vzájemného vyloučení je tedy napsat takový vstupní a výstupní kód, že celý algoritmus dodrží dvě základní podmínky.

- **Vzájemné vyloučení**: Žádné dva procesy nejsou ve své kritické sekci ve stejný okamžik.
- **Bez uzamčení**: Pokud se proces snaží dostat do své kritické sekce, pak nějaký proces, ne nutně ten stejný, se do kritické sekce dostane.

Druhá podmínka zajišťuje, že celý systém jako celek může vždy pokračovat ve vývoji. To nicméně neznamená, že nemůže dojít k hladovění *(starvation)* některého z procesů. Proto se přidává následující podmínka:

- **Bez hladovění**: Pokud se proces snaží dostat do kritické sekce, musí do ní později i vstoupit.

I když je tato podmínka mnohem silnější než podmínka bez uzamčení, stále umožňuje některým procesů násobně víckrát než procesům, které se zatím pouze chystají ke vstupu. K odstranění tohoto chování slouží čtvrtá podmínka vzájemného vyloučení:

- **Fairness**: Žádný ze vstupujících procesů nesmí vstoupit do kritické sekce před tím, než tam vstoupí procesy již čekající před nimi.

V následující kapitole se detailněji zaměříme na modelování distribuovaného vzájemného vyloučení od nejjednodušších systému po složitější. Zaměříme se na jednotlivé podmínky, které musí vzájemné vyloučení dodržet a jakým způsobem je můžeme zakomponovat do samotného algoritmu.

3.3 Model

Jak již bylo zmíněno v úvodu, první podmínka algoritmu vzájemného vyloučení zní takto:

Ve své kritické sekci může být v daný okamžik pouze jeden proces.

Jednoduchým řešením tohoto problému je použití speciální proměnné *tah*, která určí, který z procesů je na řadě se vstupem do své kritické sekce. Výsledkem je potom následující algoritmus:

```
integer tah; tah := 1;
begin
   parbegin
      process 1: begin   L1:   if tah = 2 then goto L1;
                                    critical section 1;
                                   tah   := 2;
                                   remainder of cycle 1; goto L1
                       end;
      process 2: begin   L2:   if tah = 1 then goto L2;
                                    critical section 2;
                                   tah := 1;
                                   remainder of cycle 2; goto L2
                                   end;
   parend
end;
```

Výpis 6: Model vzájemného vyloučení č. 1

Z uvedeného algoritmu je patrné, že proměnná *tah* může nabývat pouze hodnot 1 a 2. Podmínka procesu 2 pro vstup do kritické sekce je, že *tah*=2. Jediným způsobem, jak může tato proměnná nabýt tuto hodnotu, je přiřazení v procesu 1 *tah*:= 2. To je ale možné pouze po opuštění kritické sekce 1. Po tomto kroku je tedy *tah*=2, díky čemuž nemůže proces 1 znovu vstoupit do své kritické sekce a čeká, až ji opustí proces 2. Tedy že první podmínka algoritmu vzájemného vyloučení je splněna.

Zápis tohoto algoritmu pomocí Petriho sítí potvrzuje výše uvedené splnění základní podmínky vzájemného vyloučení.

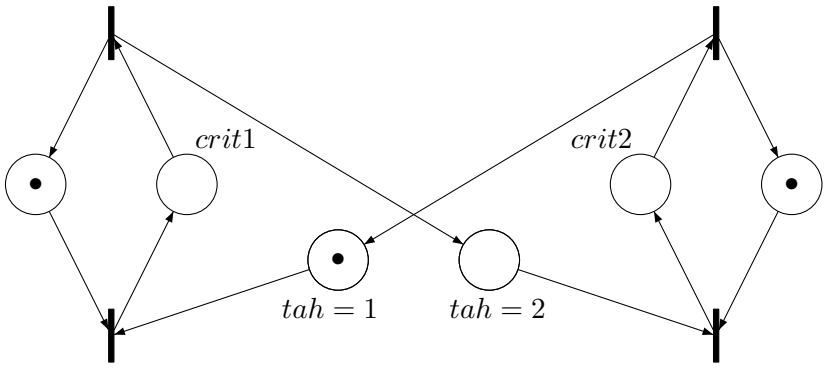


Figure 23: Model vzájemného vyloučení č. 1

Je patrné, že toto řešení dovoluje striktně danou sekvenci přístupů do kritické sekce, kdy se oba procesy pouze střídají a pokud jeden z nich přestane pracovat, celý systém přestává pracovat. Musíme tedy nalézt řešení, ve kterém výstup jednoho procesu z kritické sekce a jeho potenciální ukončení, nesmí nikdy vést k blokování druhého procesu.

V předchozím případě nám speciální proměnná tah ukazovala, který z procesů může vstoupit do své kritické sekce. Jiným způsobem řízení algoritmu vzájemného vyloučení jsou proměnné, které pouze indikují, zda je proces v kritické sekci a zda má tedy smysl, aby se do ní pokoušel dostat proces druhý. Pro další model distribuovaného algoritmu vzájemného vyloučení použijeme proměnné c1 a c2, které budou ukazovat, zda je proces v kritické sekci, či nikoliv. Procesy se těsně před vstupem do kritické sekce označí jako přítomné v kritické sekci a po vystoupení je opět označíme jako mimo svou kritickou sekci. Tím zajistíme v případě ukončení jednoho z procesů, že druhý bude moci volně vstupovat i vystupovat do své kritické sekce. Výsledný algoritmus zapsaný pseudokódem:

```
integer c1, c2;c1:= 1; c2:= 1;
begin;
   parbegin
      process 1:  begin L1:   if  c2 = 0 then goto L1;
                              c1: = 0;
                               critical  section 1;
                              c1:= 1;
                              remainder of cycle 1; goto L1
                  end;
      process 2:  begin L2:   if  c1 = 0 then goto L2;
                              c2: = 0;
                               critical  section 2;
                              c2: = 1;
                              remainder of cycle 2; goto L2
                  end;
   parend
end;
```

Výpis 7: Model vzájemného vyloučení č. 2

Na počátku mají obě proměnné c hodnotu 1, která indikuje, že jsou mimo svou kritickou sekci. Během průběhu kritické sekce 1 má c1 hodnotu 0 a proto proces 2 nemůže vejít do své kritické sekce. Problém však nastává v následující situaci. Proces 1 najde c2=1, ihned poté proces 2 zkontroluje c1, pro které stále platí c1=1. V tuto chvíli se mohou oba procesy rozhodnout, že mají volný přístup do kritické sekce, a tím porušují první podmínku. Reprezentaci tohoto algoritmu pomocí Petriho sítě znázorňuje obrázek číslo 24.

Stejně jako zápis pseudokódem i diagram Petriho sítě ukazuje problém v případě, že systém nejdříve provede akci a1, která je možná díky tomu, že oba procesy stále mají token v místech c1 a c2, ale zatím neodebere token z místa c1, protože ještě nevstupuje do kritické sekce. Pokud teď systém provede akci a2, dostane se do stavu, ve kterém mohou oba procesy vstoupit do kritické sekce a tím i porušit první podmínku vzájemného vyloučení.

Z tohoto důvodu je nutné zvolit bezpečnější řešení. Indikace vstupu do kritické sekce je jistě dobrá myšlenka, ale musíme ji trochu upravit. Možným řešením je rozšíření přípravné fáze ke vstupu do kritické sekce. V této fázi označíme proces, který chce

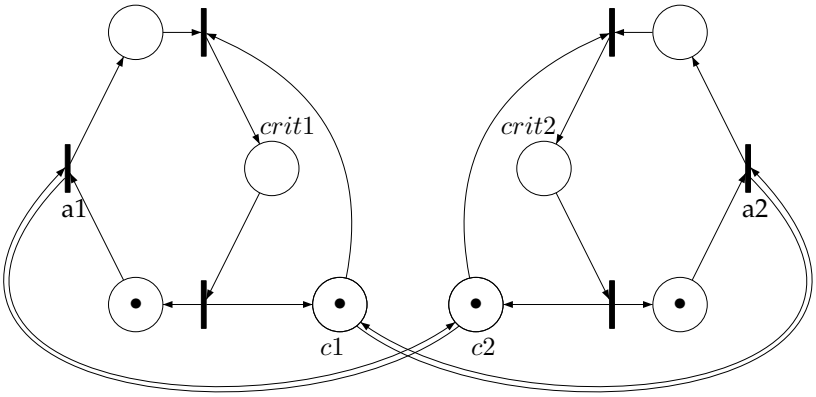


Figure 24: Model vzájemného vyloučení č. 2

vstoupit do kritické sekce jako vstupující, a teprve těsně před vstupem otestujeme, zda nám druhý proces umožní vstoupit. Tím se výsledná struktura algoritmu změní:

```
integer c1, c2;c1:= 1; c2:= 1;
begin;
   parbegin
      process 1:begin   A1: c1: = 0;
                                L1:  if  c2 = 0 then goto L1;
                                 critical   section 1;
                                c1:= 1;
                                remainder of cycle 1;  goto A1
                        end;
      process 2:begin A2: c2: = 0;
                                L2:  if  c1 = 0 then goto L2;
                                 critical   section 2;
                                c2: = 1;
                                remainder of cycle 2;  goto L2
                        end;
   parend
end;
```

Výpis 8: Opravený model vzájemného vyloučení č. 2

Pokud se zaměříme na okamžik, kdy proces 1 zjistí, že c2=1 a rozhodne se vstoupit do své kritické sekce. V tuto chvíli platí c1=0 a bude platit po celou dobu zpracování kritické sekce procesu 1. Proces 2, ve kterém platí c2=1, tj. mimo svou kritickou sekci, do ní nemůže vstoupit po dobu, kdy platí c1=0, tj. po dobu zpracovávání kritické sekce procesu 1. Diagram tohoto algoritmu vyjádření Petriho sítí se podobá předchozímu z obrázku číslo 24 a odlišuje se pouze přehozením akcí kontroly druhého procesu za označení vstupu vlastního procesu, tak jak je to vidět na obrázku číslo 25.

Problém nastává v případě, že proces 1 vejde do "přípravné" fáze pro kritickou sekci 1 a nastaví c1=0 a než ií opravdu přejde, tak proces 2 udělá to samé a nastaví c2=0. V tuto

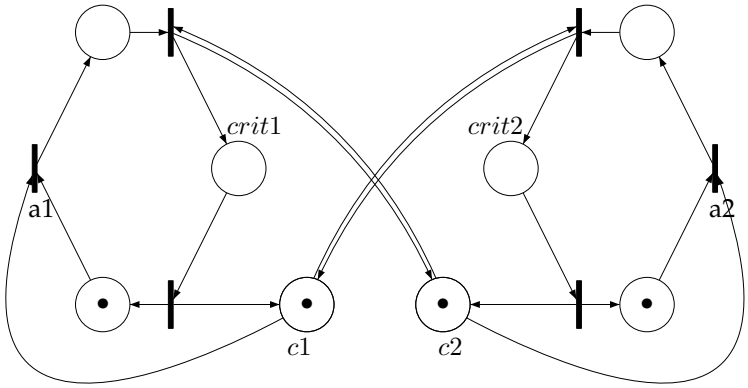


Figure 25: Opravený model vzájemného vyloučení č. 2

chvíli oba procesy vstupují do vzájemné blokace a dochází k porušení druhé podmínky vzájemného vyloučení - *bez uzamčení*.

Postup, při kterém nejprve nastavíme vlastní c, je v pořádku, chybou však je pouze spoléhat na nastavení a čekat. Proto je nutné zapracovat nějaký mechanismus, který nám bude v případě hrozícího zablokování zajišťovat návrat do počátečního stavu. Takový postup lze vyjádřit následujícím algoritmem:

```
integer c1, c2;c1:= 1; c2:= 1;
begin;
   parbegin
      process 1:begin  L1: c1: = 0;
                                        if  c2 = 0 then
                                        begin c1:=1;goto L1 end;
                                         critical  section 1;
                                        c1:= 1;
                                        remainder of cycle 1; goto L1
                       end;
      process 2:begin L2: c2: = 0;
                                         if  c1 = 0 then
                                        begin c2:=1;goto L2 end;
                                         critical  section 2;
                                        c2: = 1;
                                        remainder of cycle 2; goto L2
                       end;
   parend
end;
```

Výpis 9: Konečný model vzájemného vyloučení č. 2

Konstrukce je stejně bezpečná jako ta před ní, a pokud dojde k současnému nastavení obou c na 0, nevede tato situace ke vzájemné blokaci. Oba procesy mají nastavenu proceduru, která je v tomto případě vrací na začátek. Názorně zobrazeno pomocí Petriho sítí na obrázku číslo 26.

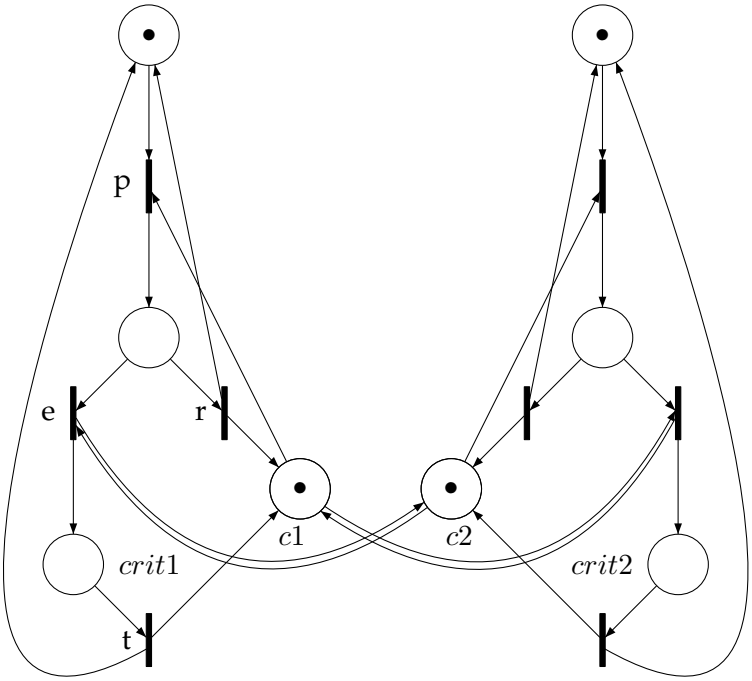


Figure 26: Konečný model vzájemného vyloučení č. 2

V uvedeném diagramu je patrné zajištění první podmínky vzájemného vyloučení tím, že provedení akce vstupu do kritické sekce **e** je přímo podmíněno přítomností tokenu v místě **c** protějšího procesu.

Dále je patrné, že v případě zastavení jednoho z procesů v místě před provedením přípravy ke vstupu do kritické sekce akcí **p**, může druhý proces volně vcházet do své kritické sekce. Tím je zajištěna podmínka živosti systému i při jednom zastaveném procesu.

Odstranění problému uváznutí v průběhu přípravy vstupu do kritické sekce nám v tomto případě obstarává konfliktní situace po provedení "přípravného" přechodu **p**.[1]. Proces se v tuto chvíli může rozhodnout vstoupit do kritické sekce pomocí akce **e**, pokud protější proces má stále token v místě **c**. Pokud ne, může provést akci **r**, při které se vrací na počátek a zároveň vrácením tokenu do vlastního místa **c** signalizuje, že protější proces může provést vlastní pokus o vstup do kritické sekce.

[1]Konflikt je vysvětlen v kapitole 2.5

Toto řešení, i když je formálně v pořádku, není zcela vyhovující. Aby celý systém fungoval v reálném prostředí, je nutné přesně nastavit rychlosti obou procesů tak, aby nedocházelo nekonečně dlouho k jejich restartování. Procesy by sice nebyly blokovány samy o sobě, ale přístup do kritické sekce ano.

Proto je nutné dále pokračovat v hledání ideálního řešení. Takové řešení jako první přinesl holandský matematik T. J. Dekker. Jde o kombinaci prvků z předchozích řešení, tedy proměnné *tah* a použití proměnných c1 a c2 k indikaci procesů, zda jsou v kritické sekci, či nikoliv.

```
integer integer c1, c2, tah;c1:= 1; c2:= 1; tah:= 1;
begin;
   parbegin
      process 1:begin  A1: c1:= 0;
                           L1:   if c2 = 0 then
                                    begin if tah = 1 then goto L1;
                                    c1:= 1;
                                    B1: if tah = 2 then goto B1;
                                          goto A1;
                                          end;
                                 critical section 1;
                                 tah:= 2; c1:= 1;
                                 remainder of cycle 1; goto A1
                    end;
      process 2:begin  A2: c2:= 0;
                           L2:   if c1 = 0 then
                                    begin if tah = 2 then goto L2;
                                    c2:= 1;
                                    B2: if tah = 1 then goto B2;
                                          goto A2;
                                          end;
                                 critical section 1;
                                 tah:= 1; c2:= 1;
                                 remainder of cycle 2; goto A2
                    end;
   parend
end;
```

Výpis 10: Dekkerův algoritmus

Je na místě si projít uvedený algoritmus, zda splňuje podmínky, které si klademe na vzájemné vyloučení. Na první pohled je zřejmé, že řešení je bezpečné a každý z procesů může vejít do své kritické sekce pouze v případě, že proměnná c druhého procesu je rovna 1. Teď je třeba ukázat, že procesy rozhodnutí o tom, který proces vstoupí do kritické sekce, nemůže být oddalováno do nekonečna. Proto se zaměříme na proměnnou *tah*.

Zápis do této proměnné může probíhat pouze na konci kritické fáze, a proto její hodnotu můžeme během tohoto rozhodování považovat za konstantní. V případě, že tah = 1, může proces 1 pouze procházet cyklem L1 s hodnotou c1 = 0 a pouze tak dlouho, dokud c2 = 0. Ale zároveň při stavu tah = 1 může proces 2 procházet pouze cyklem B2, který

implikuje c2=1. Proces 1 je tak vázán ke vstupu do své kritické sekce. Obdobný průběh má systém i v případě tah = 2.

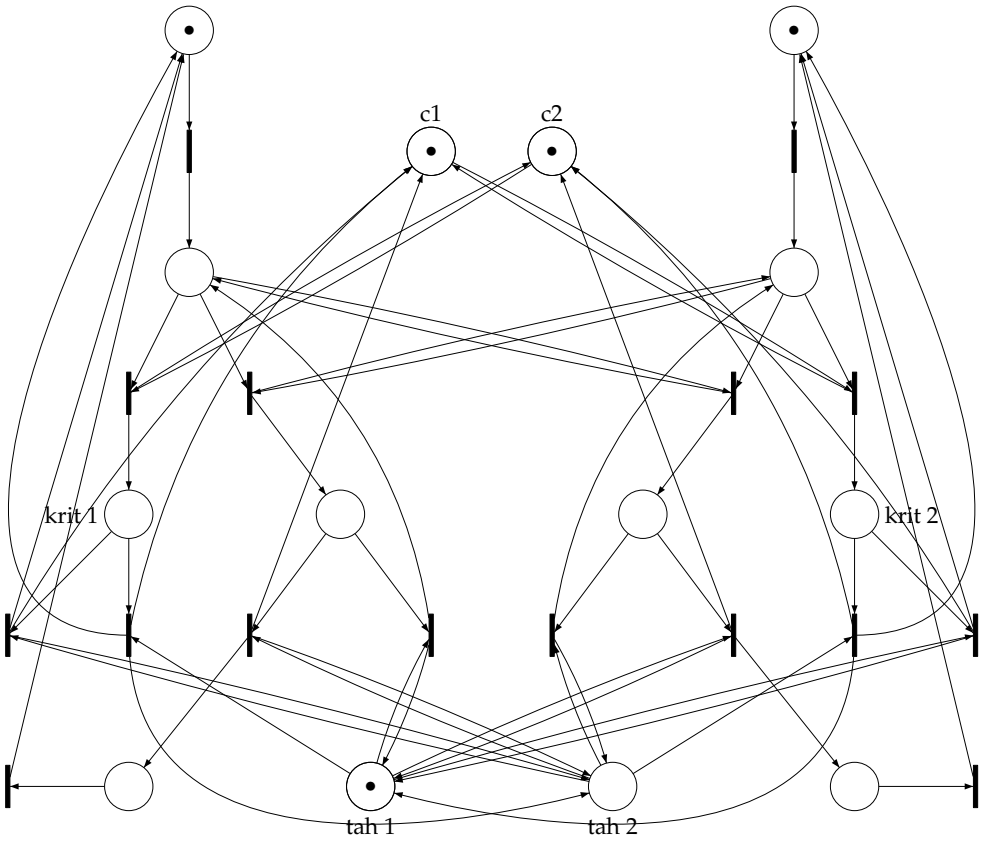


Figure 27: Dekkerův algoritmus

Závěrem prozkoumáme část algoritmu po kritické sekci. Podmínkou je, aby proces, například proces 1, který korektně vystoupil ze své kritické sekce, nezpůsobil zablokování procesu 2. Ihned po výstupu procesu 1 se vrací proměnná c = 1, tedy proces 2 má volnou cestu do své kritické sekce, a to nezávisle na aktuální hodnotě proměnné tah. Diagram tohoto algoritmu je znázorněn na obrázku číslo 27, který na druhou stranu již ukazuje jednu z nevýhod Petriho sítí při popisu složitějších, deterministických algoritmů.[2]

I když je uvedený diagram značně složitý a ne zcela přehledný, stále jsou na něm jasně patrné struktury, použité v předchozích modelech, zejména na obrázku číslo 26. Al-

[2]Nepřehlednost algoritmu z obrázku číslo 27 je do jisté míry dána i použitím nástroje LaTeX pro vytváření diagramů v této diplomové práci. Tento nástroj má pouze omezenou možnost zalamování hran a protože jsou diagramy jsou vytvářeny pomocí propojených uzlů, a tak připraveny pro případné automatické zpracování, nevyužil jsem při jejich tvorbě "pomocné" uzly.

goritmus obsahuje kontrolní místa *c1* a *c2*, která signalizují, zda se procesy snaží vstoupit do své kritické sekce. Ve spodní části diagramu můžeme pozorovat strukturu přepínání mezi *tahem 1* a *tahem 2*, která částečně snižuje možnost vstupu procesu do kritické sekce v případě opakovaných vstupů, pokud se zároveň druhý proces snaží vstoupit do své kritické sekce.

V kapitole číslo 5 na obrázku číslo 34 jsem uvedl Dekkerův algoritmus vytvořený W. Reisigem a blíže popisuji rozdíly mezi těmito dvěma přístupy.

3.4 Závěr

Jak je z této kapitoly patrné, je možné modelovat distribuovaný algoritmus různými způsoby. V přímém porovnání metody programového zápisu a reprezentace pomocí Petriho sítí je pro jednoduché systémy vhodnější použít právě grafickou reprezentaci. Její hlavní výhodou oproti programovému zápisu je možnost modelovat explicitně jednotlivé stavy systému, které v programovém zápisu musíme opsat zvlášť, a tedy přidat další informaci. Neméně důležitou vlastností grafické reprezentace je její univerzálnost a nezávislost jak na programovém jazyce, tak na jazyce obecně. Další výhodou tohoto grafického zápisu je snazší možnost analýzy systému, která je popsána v následující kapitole. Jednou z nevýhod jsou složité diagramy rozsáhlých systémů, ale i tento problém lze částečně řešit pomocí hierarchických Petriho sítí.[3]

[3]Praktickou ukázkou tohoto hierarchického řešení je skript distribuovaného algoritmu Producent konzument, vytvořený v aplikace CPN Tools a uvedený v příloze této diplomové práce.

4 Vzájemné vyloučení - analýza

4.1 Úvod

V této kapitole opustíme modelování distribuovaného algoritmu vzájemného vyloučení a zaměříme se na formální strukturální analýzu jednotlivých modelů z kapitoly předešlé. Nejdříve si ale musíme nadefinovat způsoby analýzy Petriho sítí, které obě vycházejí z grafové reprezentace. Formální teoretické definice této kapitoly jsem čerpal z [4]. [4]

Tato strukturální analýza se používá zejména v případech, kdy je PN systém neomezený (a množina dosažitelných značenía tedy i graf dosažitelnosti jsou nekonečné), a nebo v případech, kdy je PN systém sice omezený, ale množina dosažitelných značení je tak početná, že analýza grafu dosažitelnosti je neefektivní, či přímo za hranicemi možností současných výpočetních prostředků. Všechny vlastnosti Petriho sítí odvozené z její struktury jsou platné i pro všechny její systémy vzniklé přidáním libovolného počátečního značení.

Popíšeme si dvě metody:

- algebraické metody pracující s maticovou reprezentací Petriho sítí
- metody grafové reprezentace, které využívají metod redukce, metody využití zámků a pastí, studium vlastností speciálních typů PN struktur

4.2 Algebraické metody analýzy Petriho sítí

První možností je použití maticové reprezentace Petriho sítě. Využívá se incidenční matice, jejíž prvky udávají změny počtu tokenů po provedení vstupní a výstupní funkce.

Definice 4.1 *Incidenční matice PN struktury* $\langle P, T, I, O, \rangle$ *je matice* $C = C(p,t)$ *typu —P— x —T—*

$C = 0^T - I^T$,

kde $O = O(t,p)$ *a* $I = I(t,p)$ *jsou matice typu —T— x —P— reprezentující vstupní a výstupní funkci O,I.*

Prvek C(p,t) matice C udává změnu počtu tokenů v místě p po provedení přechodu t. Sloupec matice C udává změnu počtu tokenů v jednotlivých místech sítě při provedení přechodu t. Řádek matice C udává změnu značení místa p při provedení jednotlivých přechodů sítě.

Vektor, který tuto incidenční matici anuluje zleva, nazýváme *p-invariantem* PN struktury.

Definice 4.2 *P-invariantem struktury* $\langle P, T, I, O, \rangle$ *nazýváme vektor* $Y=(y_1,y_2,...,y_{|P|})^T$, $y_i \in N=0,1,2...$, *který anuluje zleva incidenční matici C*

[4] V kapitole 5.4 jsou popsány i další metody analýzy Petriho sítí, které přináší ve své monografii W. Reisig.

$Y^T \cdot C = 0.$

Petriho síť je pokryta p-invarianty, jestliže každé místo patří do nosiče nějakého p-invariantu PN struktury. Je-li Petriho síť pokryta p-invarianty, pak existuje p-invariant pokrývající celou síť. Podsystém systému $\langle P, T, I, O, M0 \rangle$ indukovaný nosičem p-invariantu P_Y se nazývá *konzervativní komponenta* tohoto systému.

Definice 4.3 *T-invariantem struktury $\langle P, T, I, O, \rangle$ nazýváme vektor $X=(x_1,x_2,...,x_{|T|})^T$, $x_i \in N=0,1,2...$, který anuluje zprava incidenční matici C*

$C \cdot X = 0.$

Petriho síť je pokryta t-invarianty, jestliže každé místo patří do nosiče nějakého t-invariantu PN struktury. Je-li Petriho síť pokryta t-invarianty, pak existuje t-invariant pokrývající celou síť. Podsystém systému $\langle P, T, I, O, M0 \rangle$ indukovaný nosičem t-invariantu T_X se nazývá *repetiční komponenta* tohoto systému.

Definice 4.4 *Předpokládejme PN struktury bez inhibičních hran. Potom PN struktura $\langle P', T', I', O', \rangle$ je duální ke struktuře $\langle P, T, I, O, \rangle$, jestliže platí:*

$P' = T \bigwedge T' = P \bigwedge I' = O \bigwedge O' = I$

Potom platí následující věta:

Nechť $\langle P', T', I', O', \rangle$, $\langle P, T, I, O, \rangle$ jsou duální sítě s incidenčními maticemi C', C. Potom platí:

$C' = C^T$,
p-invariant struktury $\langle P, T, I, O, \rangle$ je t-invariant struktury $\langle P', T', I', O', \rangle$,
t-invariant struktury $\langle P, T, I, O, \rangle$ je p-invariant struktury $\langle P', T', I', O', \rangle$,

Duální struktura vznikne z původní vzájemnou záměnou míst a přechodů a změnou orientace všech hran, beze změny jejich násobnosti. Dojde tedy i k záměně p-invariantů na t-invarianty a naopak.

V rámci analýzy distribuovaných systémů reprezentovaných Petriho sítěmi využíváme invarianty k určení některých vlastností systému.

- Existuje-li p-invariant pokrývající celou síť $\Longleftrightarrow$ PN systém je omezený.
- PN systém je živý a omezený $\Longleftrightarrow$ Existuje t-invariant pokrývající celou síť.
- Pro živý PN systém platí: Y je kladný invariant $\Longleftrightarrow$ $Y^T \cdot M_0 ¿ 0$.
- Existuje p-invariant Y s vlastností $Y^T \cdot M \neq Y^T \cdot M' \Longleftrightarrow$ Značení M' není dosažitelné ze značení M.

4.3 Grafové metody analýzy Petriho sítí

Grafové metody analýzy Petriho sítí se zabývají studiem speciálních struktur Petriho sítí, kam patří zejména pasti a zámky. V této části budeme uvažovat pouze obyčejné Petriho sítě, tedy sítě s jednoduchými hranami a bez hran inhibičních.

Zámek je podmnožina množiny míst tvořená místy, které když ztratí tokeny, je nemohou již získat.

Past je podmnožina množiny míst tvořená místy, která když tokeny získají, je již nemohou ztratit.

Na následujícím obrázku je v levé části uveden příklad zámku po provedení přechodu t1 a na pravé části je příklad pasti, která nastane po provedení přechodu t2.

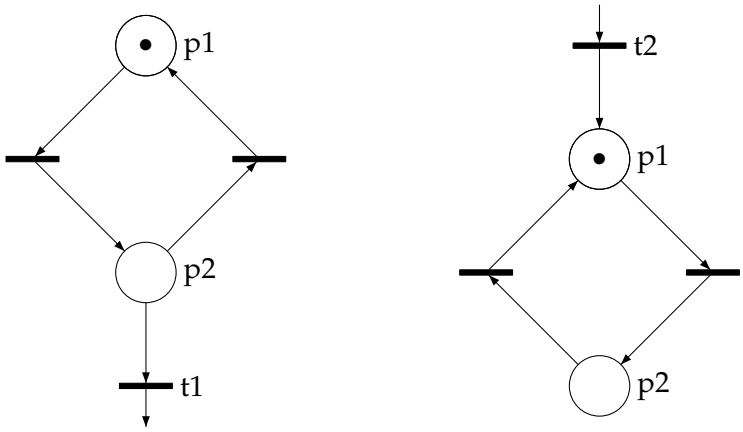


Figure 28: Příklad pasti a zámku

Zbývá dodat, že sjednocením dvou zámků je zámek a sjednocením dvou pastí je past. Můžeme tedy definovat minimální zámky a pasti Petriho sítě. Pro analýzu systému Petriho sítí je možné využít následujících vlastností:

- Obsahuje-li PN struktura zámek, který při počátečním značení neobsahuje žádný token $\Longleftrightarrow$ PN systém není živý.
- Obsahuje-li PN struktura past, která má při počátečním značení alespoň jeden token $\Longleftrightarrow$ PN systém je bez uzamčení.

4.4 Analýza modelů vzájemného vyloučení

Jakým způsobem je možné využít výše uvedených metod formální analýzy? Nejlepším způsobem demonstrace je jejich použití na modelech z předchozí kapitoly.

Jednou z důležitých vlastností algoritmu vzájemného vyloučení je nemožnost obou procesů vstoupit do kritické sekce. Jinými slovy hledáme p-invariant, který pokrývá obě

kritické sekce. Pak z jeho definice vyplývá, že žádný z procesů do něj nemůže vstoupit zároveň.

	t1	t2	t3	t4
P1	-1	1	0	0
crit1	1	-1	0	0
key	-1	1	-1	1
crit2	0	0	1	-1
P2	0	0	-1	1

Table 1: Incidenční matice sítě z obrázku číslo 29

Po anulováním zleva dostáváme tyto invarianty

	i1	i2	i3	...
1	1	1	0	...
1	0	0	1	...
0	-1	-1	1	...
1	0	-1	1	...
1	1	0	0	...

Table 2: P - invarianty matice sítě z obrázku číslo 29

Vlastní p-invariant vzájemného vyloučení je znázorněn na orázku číslo 29 silnější hranou.

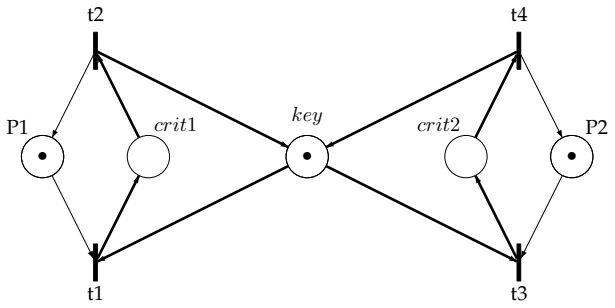


Figure 29: P-invariant algoritmu vzájemného vyloučení

Podobným způsobem zanalyzujeme další Petriho síť algoritmu vzájemného vyloučení z obrázku číslo 30, na kterém je opět vyznačen p-invariant vlastnosti vzájemného vyloučení.

Následující tabulka znázorňuje incidenční matici a některé p-invarianty této sítě.

V této tabulce odpovídá invariant **i2** invariantu znázorněnému na obrázku číslo 30.

Další strukturou, kterou lze použít pro analýzu vlastností Petriho sítí, jsou speciální síťové struktury zvané pasti. V následujícím příkladu sítě na obrázku číslo 31 použijeme

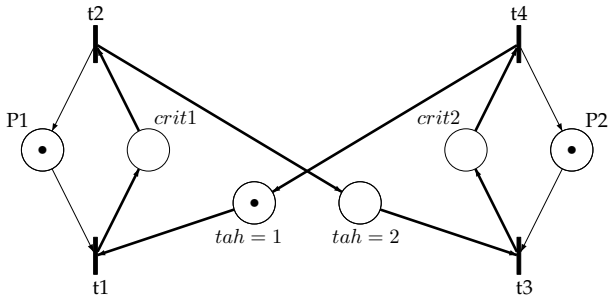


Figure 30: P-invariant modelu vzájemného vyloučení č.1

	t1	t2	t3	t4	i1	i2	i3
P1	-1	1	0	0	1	0	-1
crit1	1	-1	0	0	1	1	0
tah 1	-1	0	0	1	0	1	1
tah 2	0	1	-1	0	0	1	1
crit2	0	0	1	-1	1	1	1
P2	0	0	-1	1	1	0	0

Table 3: Incidenční matice sítě z obrázku číslo 30

pro dokázání platnosti vzájemného vyloučení postup, který popisuje W. Reisig ve své monografii Elements of Distributed Algorithms. Bližší informace k této proceduře, využívající stavových rovnic a nerovnic, naleznete v kapitole 5.4.

Musíme dokázat $\models \neg(\text{crit1} \bigwedge \text{crit1})$.

Tento vztah lze zapsat platnou síťovou nerovnicí

$C + H \leq 1$

Dále využijeme dva p-invarianty vyjádřené síťovými rovnicemi

$C + D = 1$
$E + H = 1$

a právě past, která je v síti zvýrazněna na obrázku číslo 31

$D + E \geq 1$

Odečteme nerovnici od součtu obou rovnic p-invariantů

$C + D + E + H - D - E \leq 1 + 1 - 1$

a tím jsme dokázali

$C + H \leq 1$

Samotná past je zvýrazněna silnější hranou na obrázku číslo 31.

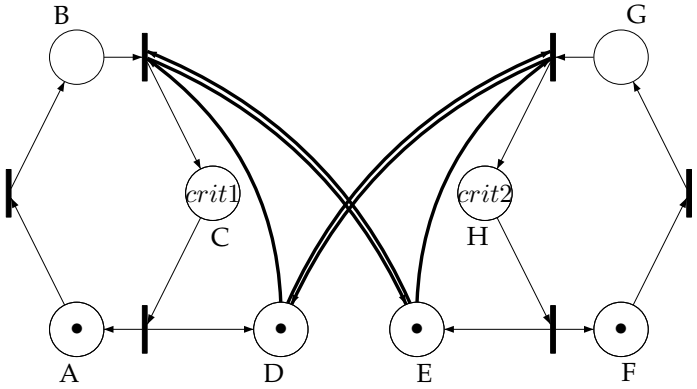


Figure 31: Past v upraveném algoritmu vzájemného vyloučení

4.5 Závěr

Se stoupající potřebou vytváření spolupracujících procesů pro řadu nejrůznějších, často velmi kritických funkcí, stoupají i nároky na jejich správný vývoj a vytváření. V případě objevení nedostatků a chyb až ve fázi prototypu bývá často nákladné a časově náročné tyto problémy odstranit a nezřídka jsou některé z těchto chyb objeveny i mnohem později. Z tohoto důvodu získává na důležitosti vytváření a analýza jejich modelů ještě v rámci vývoje.

Vytváření modelu a jeho simulace často poukáže na nové souvislosti a skutečnosti v rámci vývoje systému. Vývojový tým tak dosahuje většího porozumění než pouhým studiem vývojových dokumentů. Stejného porozumění pak mohou dosáhnout i lidé mimo vývojový tým, kteří přijdou do kontaktu s modelem. V modelu systému mohou být identifikovány jak místa podobná, která mohou být využita pro jiné, podobné situace, tak i místa potencionálně problematická, která mohou být hlouběji analyzována.

Vytvoření funkčního modelu a jeho analýza doplňuje celkovou specifikaci vytvářeného systému. Simulací modelu může být specifikace doplněna o body, které tuto simulaci znemožňují a které by vyplynuly až později u prototypu a jejichž odstranění v pozdějších fázích vývoje by bylo mnohem více nákladné či problematické. Samotná simulace systému a její analýza pak může sloužit jako prezentace budoucím uživatelům, kteří mohou blíže specifikovat svoje potřeby a požadavky, pokud je systém v modelové fázi neobsahuje. Dále se při simulaci modelů obvykle objeví řada chyb a slabých míst v návrhu. Pomocí řízené simulace, která není možná u hotového systému, je možné

projít řadou uměle vytvořených scénářů, a tak zjistit chování systému i v nestandardních podmínkách.

Díky své grafické reprezentaci jako bipartitní graf s orientovanými hranami jsou Petriho sítě intuitivně pochopitelné i lidem bez hlubších teoretických znalostí a jsou dostatečně obecné, aby šly použít na popis velkého množství různorodých systémů. Díky zápisu sítí v grafu je možné jednoduše vyjadřovat synchronizaci, vzájemné vyloučení nebo paralelní výpočty. Petriho sítě umožňují explicitní zápis stavů i událostí, což většina dalších formalismů neumožňuje. Je možné implementovat simulátory chování a formulovat metody formální analýzy. Pomocí hierarchického popisu můžeme modelovat rozsáhlé sítě pomocí menších modelů a při jejich změně není nutné příliš zasahovat do celkového modelu. Pomocí interaktivní simulace jsou výsledky znázorněny přímo v Petriho síti. V závislosti na různých potřebách vyjadřovací síly je možné používat různě modifikované Petriho sítě s dodatečnými modelovacími schopnostmi. Neméně významným přínosem je jejich přesný matematický základ, umožňující aplikovat formální metody modelování systémů s následnou verifikací kroků při zjemňování modelu.

Nevýhodami Petriho sítí je fakt, že jsou koncipovány jako plošný model, kde se sice může během modelování vyskytnout hierarchický aspekt, ale v konečném výsledku zůstává plošným modelem. Další jejich nevýhodou je neschopnost popsat dynamiku vývoje systému a jeho výjimek. Tak jak přibývá různých změn a detailů procesů, stává se systém neúměrně komplikovaný pro analýzu, a je proto nutné ho buď zobecnit, nebo použít jiné prostředky.

5 Monografie Elements of Distributed Algorithms

5.1 Úvod

W. Reisig si v monografii Elements of Distributed Algorithms dává za cíl seznámit čtenáře s možností využití Petriho sítí pro modelování distribuovaných algoritmů a jejich detailní analýzu. Za tímto účelem přináší úpravy použití Petriho sítí pro modelování jednoduchých distribuovaných algoritmů a zároveň upravuje temporální logiku za účelem jejich verifikace.

Autor se nevěnuje v této monografii celému spektru distribuovaných algoritmů, jako například N. A. Lynch v Distributed algorithms, ale vypouští real-timeové a pravděpodobnostní distribuované algoritmy. Tyto složitější struktury neodpovídají zaměření autora na základní, jednoduché modely.

Vyjímečnost této publikace spočívá v tom, že s výjimkou monografie E. Best: Semantics of Sequential and Parallel Programs (International Series in Computer Science 1996), je jediná, která používá formalismus Petriho sítí k modelování souběžnosti distribuovaných systémů.

Autor Wolfgang Reisig rozdělil knihu do čtyř hlavních částí. V první části je čtenář seznámen se základní teorií Petriho sítí a je mu představeno několik jednoduchých algoritmů.

V druhé části autor rozvíjí základní systémy o systémy popsané Petriho sítěmi vyšší úrovně a se složitějšími modely již zmíněných algoritmů.

Třetí část se věnuje formální analýze distribuovaných algoritmů. Autor zde přináší několik technik sloužících k analýze, zejména formule temporální logiky a využití invariantů.

V závěrečné části pak autor analyzuje algoritmy, které uvedl v předchozích kapitolách a dokazuje, že skutečně nabývají požadovaných vlastností.

5.2 Základní systémové modely

V první části knihy se autor věnuje různým formalismům, pomocí nichž je možné popsat distribuovaný algoritmus. Jako průvodce mu slouží jednoduchý algoritmus producent - konzument. Autor vysvětluje v čem má využití Petriho sítí přednosti před ostatními formalismy a také se věnuje obecné charakteristice vlastností Petriho sítí, zejména s ohledem na využití pro modelování distribuovaných algoritmů. V této části je také srovnání s jinými monografiemi, zejména na kolik mají stejné zaměření a v čem se liší.

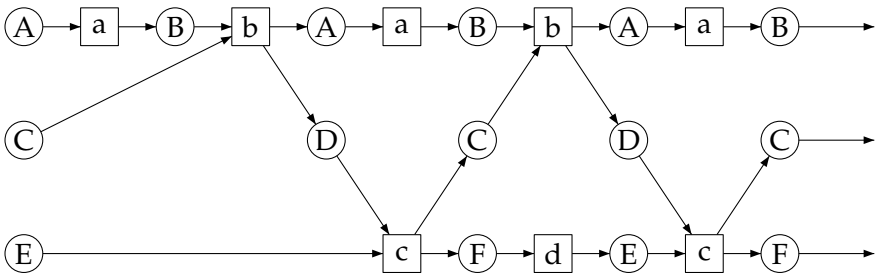
Po tomto úvodu již následuje vysvětlení a formální teorie Petriho sítí[5] jako základního kamene pro modelování distribuovaných algoritmů. Autor definuje strukturu sítě, její prvky a různé speciální struktury, které se mohou objevit.

Dále autor uvádí dva různé pohledy na dynamiku Petriho sítí. Nejprve formálně definuje přechody mezi stavy jako konečný či nekonečný sekvenční (*interleaved*) běh, tedy jako pouhý sled stavů.

[5]Tato teorie je obsažena v kapitolách 2.3 a 2.6

Jako druhý pohled definuje běh souběžný (*concurent*), který pak již přináší explicitní znázornění procesů, které mohou souběžně probíhat. Autor tohoto pohledu často využívá v pozdějších kapitolách, a proto mu věnuje zvýšenou pozornost. Na tomto souběžném běhu dále demonstruje využití Petriho sítí nejen pro modelování samotného systému, ale jak je patrné z obrázku číslo 32, je možné Petriho sítě využít i na znázornění a analýzu možného průběhu algoritmu. V tomto znázornění můžeme například analyzovat dostupnost stavů nebo možnost uváznutí.

Na následujícím obrázku je maximální souběžná sekvence algoritmu producent - konzument. Pro větší srozumitelnost jsou jednotlivé stavy a akce pojmenovány.



A: připraven k výrobě
B: připraven k dodání
C: prázdný sklad
D: plný sklad
E: připraven k odebrání
F: připraven ke spotřebě

a: výroba
b: dodání
c: odebrání
d: spotřeba

Figure 32: Maximální jedinečná souběžná sekvence systému producent - konzument z obrázku číslo 2

Následující dvě kapitoly se zabývají vlastnostmi, které standardní teorie Petriho sítí úplně nepokrývá, a které autor zavádí pro řízení chování algoritmů. První z nově zavedených vlastností je progrese. Na příklad v systému producent - konzument je žádoucí umožnit pouze konečnou sekvenci akcí *odběr* a *spotřeba*. Toho je dosaženo tím, že v určitém okamžiku se nespustí akcw *výroba*. Jak sekvenční, tak souběžný systém tohoto systému pak nesmí podporovat progresi akce výroba, ale všech ostatních ano. Autor tuto vlastnost formálně definuje právě v závislosti na průběhu algoritmu a zavadí speciální značku q právě pro ty procesy, u kterých nechce podpořit progresi.

Další z nových vlastností Petriho sítí, které autor přináší, je *fairness* přechodu. Aby přechod dodržel tuto vlastnost, nesmí být v systému nekonečněkrát umožněn a pouze konečněkrát proveden. I tato vlastnost je formálně definována a pomocí jednoduchého příkladu demonstrována. Pro tuto vlastnost autor zavádí značku φ.

Závěrem úvodní teoretické části autor definuje speciální strukturu Petriho sítě zvanou základní systémová síť. Aby Petriho síť mohla být definována jako základní systémová síť, musí splňovat následující podmínky:

- Síť má počáteční stav
- Přechody sítě buď podporují, nebo nepodporují progresi
- Některé přechody, podporující progresi, mohou dodržovat fairness

Tuto strukturu základních systémových sítí (*es-net*) využívá autor v průběhu celé knihy.

Druhá část úvodní kapitoly obsahuje modely základních distribuovaných algoritmů. Autor se v této části omezuje na použití bezpečných (tedy 1- omezených) Petriho sítí bez inhibičních hran.

V úvodu této kapitoly se autor vrací k systému producent - konzument a rozšiřuje jeho model nejprve o druhý sekvenční sklad a následně i o druhý paralelní sklad. Zároveň předkládá výhody a nevýhody obou řešení, aby nakonec představil třetí model s deterministicky řízeným přístupem do paralelního skladu jako na této úrovni nejlepší řešení problému. Jednotlivé vlastnosti systémů pak demonstruje i na jejich souběžných bězích.

Dalším algoritmem, který autor představuje, je populární a známý algoritmus hladových filozofů. Jak je zvykem i v dalších kapitolách je každý nově představený algoritmus krátce uveden a jeho problém popsán. Jenom vyjímečně je v této kapitole řešen determinismus, a proto veškeré průběhy algoritmů slouží pouze autorovi, aby čtenáře dostal do situace, kterou mu chce popsat a zdůraznit případný problém. I v případě hladových filozofů autor postupuje stejným způsobem. Nejprve představuje jednoduchý zápis souběžného běhu. Na příkladu algoritmu z kapitoly 2.6 uvádím takovýto zkrácený zápis na obrázku číslo 33.

Autor se na základě takto zapsaného průběhu tohoto algoritmu zabývá otázkou, zda je možné najít nějakou "spravedlivou" sekvenci střídání filozofů, a pokud ano, kolik takových průběhů existuje.

Následuje algoritmus asynchronního zásobníku. Autor předvádí Petriho síť jednoduchého modulu, který se skládá do sekvence za sebou, a tvoří tak asynchronní zásobník. Každý z modulů uchovává jednu informaci. Pro tyto moduly jsou definovány funkce *push* a *pop*. Funkce push posílá novou hodnotu směrem ke dnu zásobníku a funkce pop pak vytahuje uloženou hodnotu směrem k vrcholu.

Následující kapitola představuje *crosstalk* algoritmus. Tento algoritmus má za úkol řídit předávání zpráv mezi agenty, které dodržují "kruhový" pracovní cyklus. To znamená, že pokud odesilatel odešle zprávu ve svém i-tém kroku, je nutné, aby ji i příjemce obdržel ve svém i-tém kroku. Autor uvádí hned několik variant tohoto algoritmu. Jako první představuje velice jednoduchý algoritmus, skládající se ze dvou procesů, kdy jeden z procesů je odesílatel a druhý příjemce. Následuje rozšíření systému tak, aby oba procesy mohly zastávat roli jak příjemce, tak odesílatele. Petriho síť tohoto algoritmu však obsahuje nebezpečí uváznutí ve chvíli, kdy se oba procesy rozhodnou odeslat zprávu. Proto autor stávající konstrukci ještě více rozšíří a tento problém odstraní. Na základě prozkoumání Petriho sítě souběžného běhu se však ukazuje, že i v tomto systému existují procesy, ze kterých není možný další posun. Proto je nutné systém dále upřesnit, a tak je vytvořen dostatečně dokonalý systém bez uváznutí, což je také následně demonstrováno Petriho sítí jeho maximálního souběžného běhu.

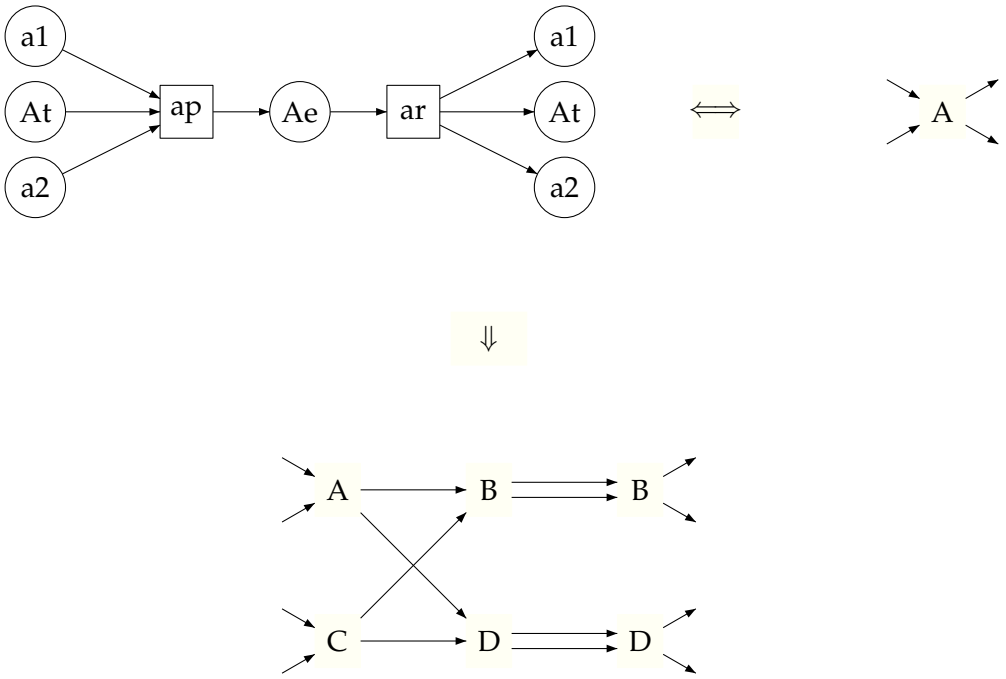


Figure 33: Průběh algoritmu hladových filozofů zapsaný zkrácenou formou

V závěru této části knihy otevírá autor kapitolu vzájemného vyloučení. Tomuto tématu se věnuje i moje diplomová práce v kapitole 2.5, ale především v kapitole 3. Autor z počátku vychází stejně jako já ze stejného zdroje [5], kde publikuje stejné (až na případný isomorfismus), nebo obdobné (lišící se o substituci místa a stavu) Petriho sítě, aby posléze představil nové algoritmy, které nejsou zahrnuty v mé diplomové práci. Sem patří zejména Petersonův algoritmus, Owicki-Lamportův algoritmus a asymetrická varianta vzájemného vyloučení.

Zvláštní srovnání si zaslouží Dekkerův algoritmus. Petriho síť, kterou jsem navrhl ve své diplomové práci a je zobrazena na obrázku číslo 27 ,se liší od algoritmu, který uvádí autor ve své monografii. Pro ilustraci uvádím Reisigovu reprezentaci Dekkerova algoritmu na následujícím obrázku číslo 34.

Společnými prvky obou reprezentací je zavedení stavů, které hlídají, zda se druhý proces chystá vstoupit do kritické fáze, a stavů, které řídí vstup pomocí sdíleného tokenu, v případě pokusu obou procesů v přístupu do kritické sekce. Narozdíl od mého modelu umožňuje Resigův model vstup do kritické sekce i v případě, že místa *finished* neobsahují token a to v závislosti na stavu míst *at* a *pending*. Další odlišnost je v přístupu k podmínce stanovené E. W. Dijsktrou na algoritmus vzájemného vyloučení, a která zní: "Pokud se nějaký proces ukončí korektně mimo svojí kritickou sekci, nesmí toto vést k potenciálnímu blokování druhého procesu". V mém řešení je tato podmínka dosažena přímým přechodem z kritické sekce do počátečního stavu s vrácením tokenu do kon-

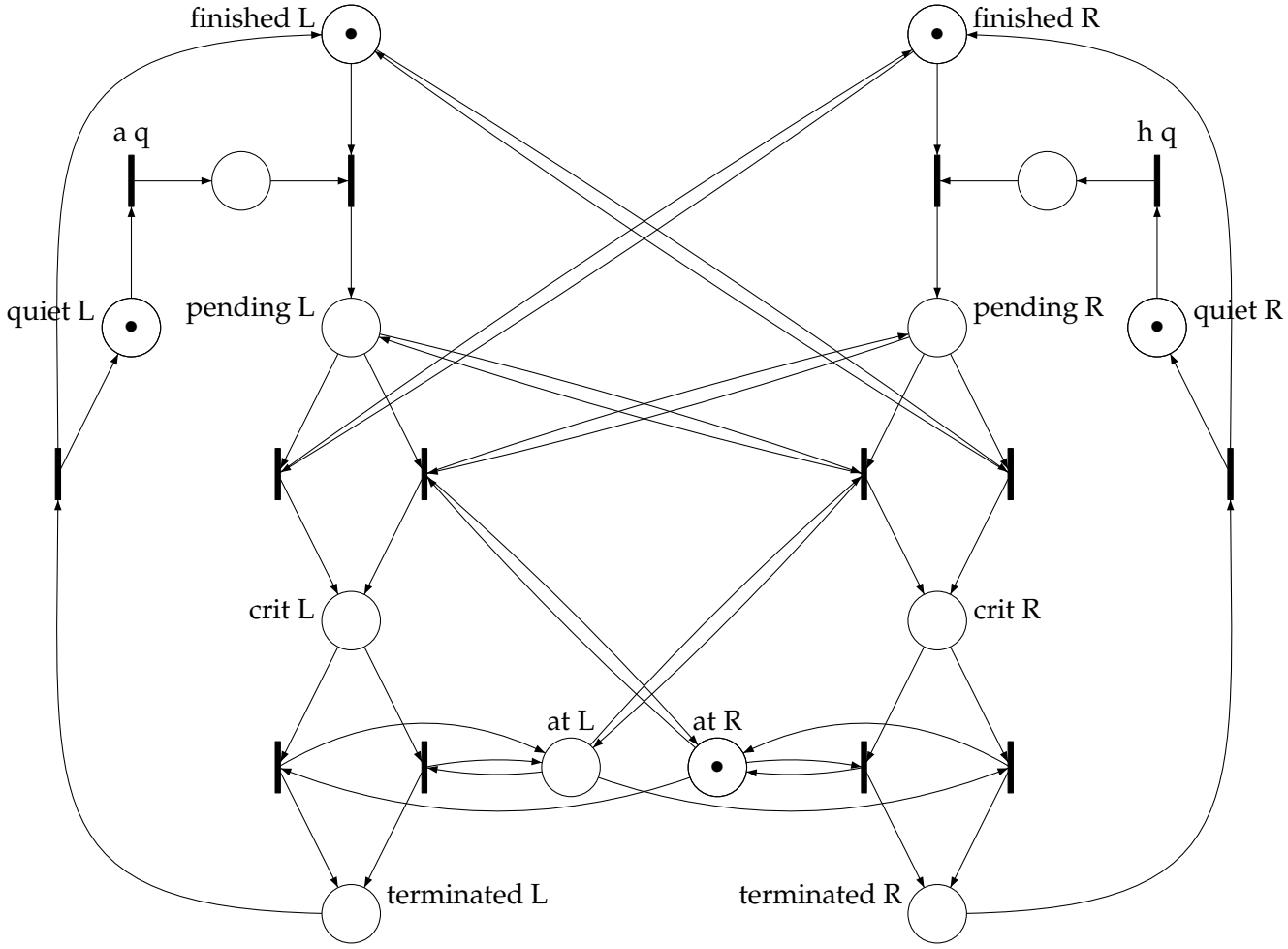


Figure 34: Dekkerův algoritmus podle W. Reisiga

trolního bodu. W. Reisig řeší tento požadavek tak, že označí přechody *a* a *h* jako "klidové" (nepodporující progresi), a tak je algoritmus vždy ukončen ve stavu *quiet*, který nebrání druhému procesu ve vstupu do kritické sekce. V případě, že by nebyly uvedené procesy označené jako klidové, mohlo by dojít k ukončení práce jednoho z procesů ve stavu *terminated*, a díky tomu k nemožnosti vstupu druhého procesu do své kritické sekce.

5.3 Pokročilé systémové modely

Ve druhé části své monografie rozvíjí W. Reisig možnosti formalismu Petriho sítí o integraci datových struktur a zavádí novou strukturu - systémové sítě. Na tyto sítě lze pohlížet ze dvou odlišných úhlů. V první řadě se jedná o zobecnění základních systémů. Tokeny nejsou již pouhé černé tečky, ale mohou znázorňovat jakákoliv data. Druhá cesta, jak tyto sítě chápat, je zkrácená reprezentace základních sítí. Pomocí metod grafové substituce můžeme spojit místa i přechody, které mají stejný charakter a funkci a tím pádem model zjednodušit a zpřehlednit, aniž bychom ztratili vlastnosti, které od těchto systémů očekáváme.

V úvodu této části se autor vrací ke dvěma algoritmům z první části, konkrétně k algoritmu producent/konzument a algoritmu hladových filozofů a na nich demonstruje nový přístup k modelování pomocí konkrétních datových struktur namísto obyčejných tokenů. Postup takovéto substituce je názorně vysvětlen řadou postupných kroků, přičemž každý z nich je demonstrován příslušným diagramem. Úvod druhé kapitoly uzavírá algoritmus Erathostenova síta, který svým zaměřením přímo vybízí k distribuovanému výpočtu, protože odstraňování čísel, která nejsou prvočísly, může být prováděno v jakémkoliv pořadí i paralelně. Následující obrázek představuje diagram tohoto algoritmu pro n- čísel a je již reprezentován zjednodušeným zápisem.

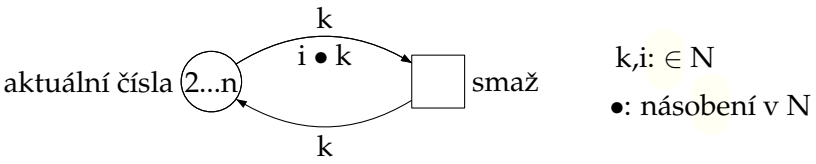


Figure 35: Distribuovaný algoritmus Eratosthenova síta

Přechod *smaž* v tomto algoritmu může nastat pouze tehdy, pokud nějaké číslo k a nějaký jeho násobek i • k je obsažen v aktuálních číslech. Pak je k a i • k smazáno a samostatné k se vrací do aktuálních čísel.

Následuje formální rozšíření definicí potřebných pojmů těchto *systémových* sítí o prvky substituce míst a přechodů a definice jejich sekvenčních a souběžných běhů. Pro další

zjednodušení těchto systémových sítí definuje autor nově reprezentaci systémových sítí pomocí termů a funkcí. Jednotlivé hrany, tedy podmínky provedení přechodu, jsou popsány pouze termy, kterým je následně dodán význam. Z tohoto důvodu je možné jedné struktuře Petriho sítě nadefinovat různé podmínky přechodů, a tedy i různé chování výsledného algoritmu. Příklad zápisu pomocí těchto termů a funkcí představuje předchozí obrázek 35.

Pro potřeby rozhodování v závislosti na datech představuje autor přechodovou stráž *(transition guard)*, strukturu míst a přechodů schopnou řídit algoritmus v závislosti na procházejících datech. Závěrem teoretické části druhé části monografie seznamuje autor čtenáře se systémovým schématem. Tato schémata mohou popisovat různé sítě stejné struktury, lišící se pouze v definovaných doménách, konstantách a funkcích, které se mohou měnit. Systémová schémata tak vytvářejí sady systémových sítí.

V následující sekci případových studií se opět vrací k algoritmům z úvodní kapitoly a modeluje je s ohledem na nově definovanou strukturu systémových sítí. Čtenář má možnost se seznámit s dříve popsaným algoritmem producent - konzument, který je nyní rozšířen o sekvenční a paralelní sklad, respektive n skladových buněk. Další algoritmus, ke kterému se autor vrací, aby jej rozšířil, jsou hladoví filozofové. Autor k jejich systémovému schématu připojuje místo, ve kterém jsou přednastaveny jednotlivé dvojice filozofů v pořadí, ve kterém mají začít jíst. Na tomto příkladu je demonstrováno zapojení funkce priority do distribuovaného algoritmu jako další řídící funkce.

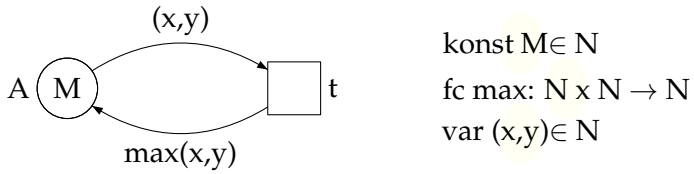


Figure 36: Distribuovaný algoritmus hledání maximální hodnoty

Dalším odvětvím,ve kterém je možné uplatnit distribuovaný přístup, je distribuované programování s omezujícími podmínkami *(Distributed constraint programming)*. Nalezení řešení problému pomocí tohoto programování spočívá v postupném vyřazování kandidátů na řešení z velké množiny možných kandidátů. Protože mohou být prvky množiny omezovány nezávisle na sobě, je souběžné spouštění pomocí distribuovaných algoritmů často využívané.

Již zmiňovaný algoritmus Eratosthenova síta z obrázku 35 je takovým příkladem programování s omezujícími podmínkami. Dalším příkladem je distribuované hledání maximální hodnoty z obrázku 36. Autor předkládá čtenáři další jednoduché distribuované algoritmy, mezi nimiž jsou distribuované třídění, distribuované hledání nejkratší cesty a další.

Jako varianta algoritmu vzájemného vyloučení jsou dále představeny algoritmy řešící exkluzivní zápis a souběžné čtení. Jak má W. Reisig ve zvyku jsou čtenáři postupně předkládány různá řešení od toho nejjednoduššího, ale nespravedlivého, k složitějším algoritmům, které již mají lepší vlastnosti. Stejným způsobem je postupováno i v případě algoritmu distribuovaného seřazení. Závěr těchto případových studií tvoří návrat k algoritmu vzájemného vyloučení, který je rozšířen na samostabilizující se vzájemné vyloučení. Tento princip spočívá v tom, že pokud v průběhu zpracování jednotlivých procesů dojde ke stavům, ve kterých není zajištěno vzájemné vyloučení, je algoritmus sám schopen se opět dostat do stavu, který vzájemné vyloučení opět dodržuje.

Následující kapitola představuje čtenáři komunikační algoritmy. Ty jsou další velkou a často používanou skupinu distribuovaných algoritmů. Autor se v této části zaměřuje zejména na dosažení spolehlivosti předávaných zpráv, případně jejich pořadí.

Jako první je představen princip předávání potvrzení zpráv. Jak již bylo několikrát řečeno, postup popisu řešení spočívá v přechodu od nejjednodušího algoritmu, který se sestává pouze z kruhové sítě s odesílatelem na jedné straně a příjemcem na druhé straně, a který se zablokuje ihned po té, co dojde ke ztrátě zprávy. Z tohoto důvodu je do tohoto algoritmu přidán prvek, který zajišťuje zaslání kopie v případě ztráty původní zprávy. V tomto algoritmu ale nastává problém s identifikací zpráv, protože není zřejmé, zda se jedná o originál či kopii, a proto je nutné tento algoritmus rozšířit i o identifikaci zprávy. Tímto způsobem autor modeluje celý systém, dokud nedosáhne požadovaných vlastností v přiměřené míře složitosti algoritmu.

Takto navržený algoritmus má striktně udán sled zpráv, tedy odesílatel zašle novou zprávu i+1 až poté, co obdrží potvrzení o doručení zprávy i. Proto autor představuje další možnosti předávání zpráv a to pomocí pohyblivého okna. Odesílatel pak může poslat všechny zprávy, které mají svůj identifikátor mezi dvěma indexy. Stejným způsobem je řešeno i přijímání potvrzení. Autor čtenáři předkládá jak systém s neohraničenými indexy, tak s indexy ohraničenými. V závěru této kapitoly se pak autor zaobírá opět obecnějšími diagramy předávání zpráv mezi jednotlivými sousedy v distribuované síti.

Tématem těchto distribuovaných sítí, zejména pak distribuovanými algoritmy, které v nich pracují, se autor zabývá v poslední kapitole Pokročilých systémových modelů. Autor definuje *síťové algoritmy*, které nemají pouze fungovat v rámci jedné fixní sítě, ale jedná se spíše schéma algoritmů, které mohou pracovat v celých různých třídách sítí.

Základní myšlenkou je zobrazení lokálních algoritmů v obecném zápise s explicitním vyjádřením předávaných zpráv. Z tohoto důvodu autor přináší dva nové principy do modelování takovýchto systémů. Prvním z nich je ohodnocení vstupních hran každého přechodu n-ticí proměnných, často označovaných x. Ohodnocení každé příchozí hrany přechodu má na prvním místě n-tice vždy stejnou proměnnou. Druhým novým prvkem je předávání zpráv. Každá zpráva je zobrazena jako n-tice (x_1,....,x_n), často pro n = 2, kde x_1 je příjemcem zprávy, x_2 je odesilatelem zprávy a x_3...x_n mohou obsahovat jakákoliv data.

Po tomto krátkém teoretickém úvodu představuje autor čtenáři algoritmy, které dosud nepopisoval v předchozích částech knihy. Prvním z těchto nových síťových algoritmů je algoritmus výběru vedoucího. Tento obecně známý algoritmus je krátce popsán

a na diagramu vysvětlen. Obsáhlejší popis tohoto algoritmu obsahuje kapitola Výběr vedoucího této diplomové práce.

Tento algoritmus je ukončen s tím, že každý proces zná jméno vedoucího. V okamžiku, kdy každý proces bude navíc znát i vzdálenost k vedoucímu a souseda, který je k němu blíž, bude moci komunikovat s vedoucím právě přes tohoto souseda. Trasy mezi těmito sousedy pak tvoří minimální kostru grafu. Tohoto si všímá autor a představuje tento algoritmus jako variantu algoritmu hledání vedoucího. Na tomto příkladu je zřejmý rozdíl mezi dvěma algoritmy, které mají sice stejnou strukturu, ale díky jinému označení pomocí termů a funkcí jiný průběh.

Návratem k předávání zpráv v distribuované síti je Echo algoritmus. Tento jednoduchý algoritmus vysílá zprávy do celé sítě a neukončí se do té doby, dokud neobdrží potvrzení od všech ostatních uzlů.

Autor pokračuje rozšířením algoritmu vzájemného vyloučení do sítí. V těch rozeznává lokální a globální vzájemné vyloučení. Globální vzájemné vyloučení zahrnuje exkluzivní přístup do kritické sekce pouze jedinému algoritmu v celé síti, v případě lokálního vzájemného vyloučení stačí pouze exkluzivní přístup mezi sousedícími procesy. Příkladem tohoto lokálního vzájemného vyloučení je například algoritmus hladových filozofů, kdy stačí dodržet vzájemné vyloučení mezi sousedními filozofy. Pro oba tyto přístupy autor navrhuje distribuované algoritmy s krátkým popisem dosažitelných stavů a funkcí.

Takto bez nějakého dlouhého popisu se autor dotýká také problému konsensu v sítích, kdy přináší několik variant možných řešení, opět bez nějakého hlubšího pozadí. Jeden z těchto algoritmů představuje obrázek číslo 37.

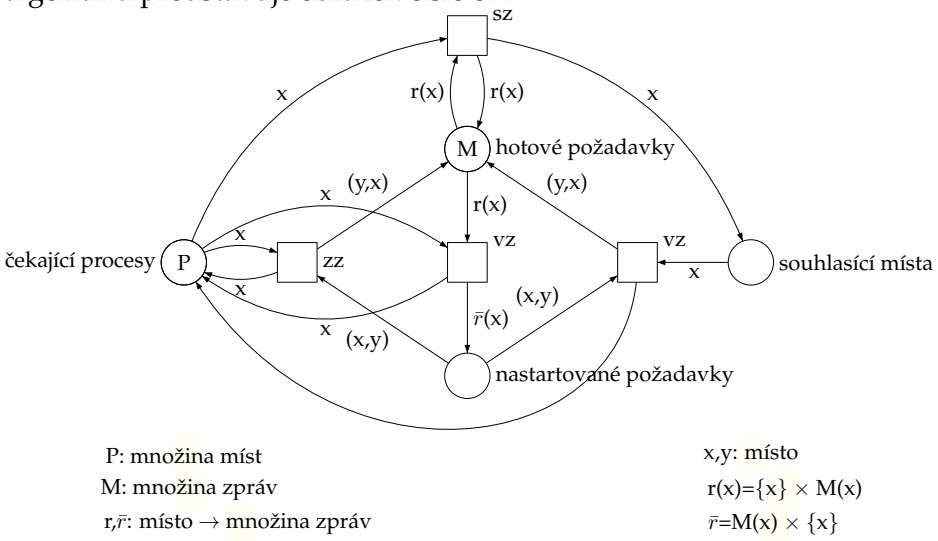


Figure 37: Distribuovaný algoritmus konsensu

Na začátku je každé místo čekající a každá zpráva (požadavek) hotová. Pomocí akce *zz* může jakékoliv místo x zaslat svému sousedu y zprávu (x,y). Po obdržení zprávy si ji místo x přečte a vrátí ji odesilateli y pomocí některé akce vz. Každá z těchto akcí změní stav místa x na čekající. Závěrem může místo x pomocí akce sz změnit na souh-

lasící a všechny své zprávy r(x) označit jako hotové. Tento algoritmus nezajišťuje stav, ve kterém všechna místa jsou souhlasící, ale zajišťuje stability, protože algoritmus skončí pouze tehdy a jen tehdy, když všechna místa souhlasí.

Poslední algoritmus, který je v této sekci představen, je algoritmus distribuovaného sebestabilizování. Tento algoritmus je nutný v případě několika procesů, které zpracovávají stejné úkoly a komunikují s prostředím. Tyto úkoly jsou průběžně zpracovávány a zároveň z prostředí přicházejí nové. Často se stává, že pracovní náplň jednotlivých procesů je nerovnoměrná. Úkolem algoritmu tedy je , aby v případě přetížení jednoho z procesů převedl část pracovní náplně na proces, který právě teď nemá tolik úkolů.

Touto kapitolou autor opouští modelování jednotlivých algoritmů spolu s definováním teoretického pozadí a přechází do druhé části knihy, která se zabývá možnostmi analýzy těchto systémů.

5.4 Analýza základních systémových modelů

V této části opouští autor modelování distribuovaných systémů a přistupuje k metodám jejich analýzy, kterou právě reprezentace pomocí Petriho sítí umožňuje. Analýzou je myšleno zobrazení určitých vlastností systému. Tyto vlastnosti buď platí, nebo neplatí. Příkladem takové vlastnosti v algoritmu vzájemného vyloučení je umožnění vstupu do kritické sekce pouze jednomu procesu.

Úvodem této kapitoly autor představuje koncept využití temporální logiky pro vytvoření logických formulí, pomocí nichž lze jednotlivé vlastnosti Petriho sítí popsat. Tato temporální logika není přestavena a použita v její plné síle, slouží spíše jako nástroj pro vytvoření intuitivních výroků, které mohou ulehčit pochopení popisovaných systémů. [6]

Tyto jednoduché formule se odvozují přímo ze struktury Petriho sítě a z nich jsou následně odvozována složitější pravidla. Formule popisují stavy, které v příslušné Petriho síti mohou nebo nemohou nastat. Příklad tohoto využití převodu Petriho sítě na logické formule demonstrují pravidla sítě z obrázku číslo 38.

V uvedené Petriho síti platí například pravidlo:

- $\models (B \rightarrow C) \bigwedge (A \rightarrow \neg C)$

To znamená že ve všech dosažitelných stavech této sítě platí, pokud je token v místě B, je i v místě C a zároveň pokud je token v místě A, není v místě C. Použitím těchto formulí v analýze Petriho sítí můžeme přímo definovat jejich některé vlastnosti, jako je například neexistující kontaktní situace nebo systém bez uzamčení.

Stavové vlastnosti Petriho sítí mohou být potvrzeny také pomocí takzvaných *síťových rovnic a nerovnic*. Tyto rovnice a nerovnice můžeme vytvářet přímo ze statické struktury jakékoliv Petriho sítě, kde je každé místo sítě bráno jako proměnná nabývající hodnot 0 a 1, a každý stav sítě je reprezentován charakteristickou funkcí. Ta přiřazuje každému místu stavu hodnotu 1 pokud ve stavu obsahuje token a 0 pokud token neobsahuje. Síť z obrázku číslo 38 obsahuje následující platnou rovnici a nerovnici:

[6] Pochopení tohoto principu od čtenáře vyžaduje alespoň základní znalosti teorie logiky.

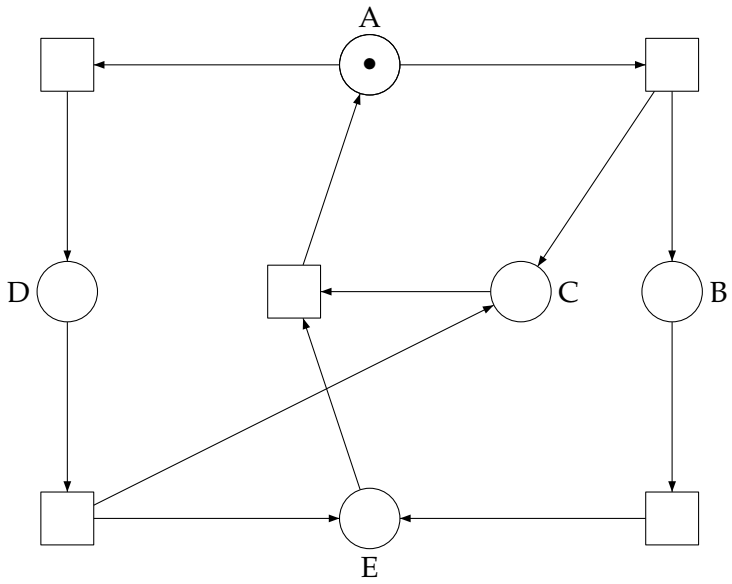


Figure 38: Distribuovaný algoritmus

- B - C + E = 0
- A + B + C + D + E $\geq$ 1

Tyto síťové rovnice a nerovnice můžeme také využít v analýze Petriho sítí, protože samy o sobě implikují platné vlastnosti systému. Například rovnice z předchozího příkladu implikuje $\models B \rightarrow C$.

Další prvek analýzy, který autor představuje a definuje, jsou p-invarianty. Vytvoření p-invariantů z incidenční matice popisuje a definuje Reisig podobně jako v kapitole 4 této diplomové práce, která čerpá z [4]. Zajímavější je jejich využití jako síťových rovnic.

Síť z obrázku číslo 38 obsahuje následující p-invarianty:

- A + C + D = 1
- A + B + D + E = 1
- 2A + B + C + 2D + E = 2
- B - C + E = 0

Takto zapsané p-invarianty je možné snadno zobrazit. Na obrázku číslo 39 je tučnou čárou znázorněn první invariant z předchozího příkladu.

Stejně jako v předchozích kapitolách, i zde po seznámení s novou teorií, přináší autor několik příkladových studií již dříve namodelovaných algoritmů. V algoritmu producent - konzument se dvěma sekvenčními sklady zobrazí všechny invarianty pro jednotlivé procesy. Stejným způsobem zanalyzuje i deterministický distribuovaný algoritmus producent - konzument s paralelními sklady.

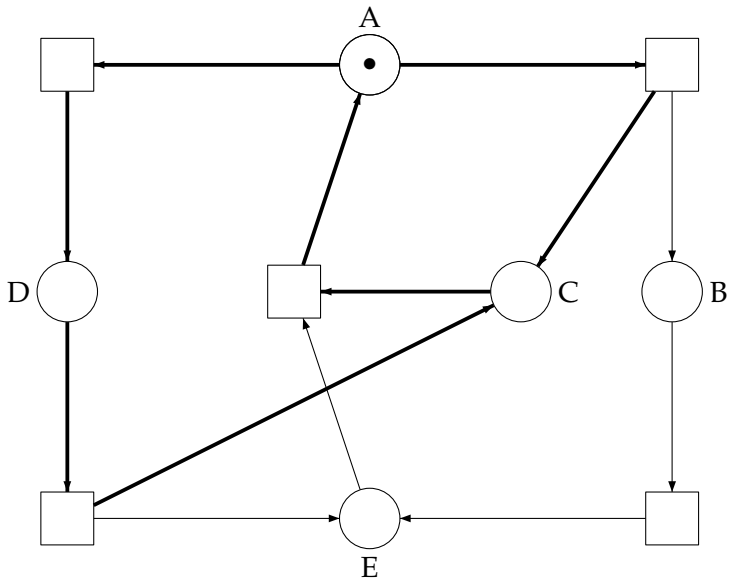


Figure 39: Invariant A + C + D = 1

Další skupinou algoritmů, na kterých může autor ukázat výhody použití invariantů, jsou algoritmy vzájemného vyloučení. Pomocí analýzy těchto systémů dokazuje platnost právě vzájemného vyloučení, tedy stavu, kdy do své kritické sekce může vstoupit pouze jeden z procesů. Tyto analýzy jsou blíže popsány v této diplomové práci v kapitole 4. Zároveň ale v této kapitole musí konstatovat "slabinu" použití p-invariantů. Tyto invarianty není možné použít v Petriho sítích, které obsahují smyčky. Smyčku není možné zaznamenat v incidenční matici a proto také není možné využití p-invariantů.Příklad takovéto sítě je znázorněn na obrázku číslo 26.

Z tohoto důvodu autor, jako další prvek analýzy, definuje využití pastí, a to konkrétně pastí inicializovaných. Pastmi Reisig rozumí takové skupiny míst, jejichž výstupní přechody jsou podmnožinou přechodů vstupních. Tak jako invariant implikuje síťovou rovnost, past naopak implikuje síťovou nerovnost.

Na následujícím obrázku číslo 40 je zvýrazněna past A,D.

K analýze systému můžeme využít i vzájemnou kombinaci invariantů a pastí. V případě sítě z obrázku číslo 40 chceme dokázat $\models B \rightarrow D$.

$B \leq D$

Tato síť obsahuje invariant

$A + B = 1$

a zároveň výše jmenovanou past A,D

$A + D \geq 1$

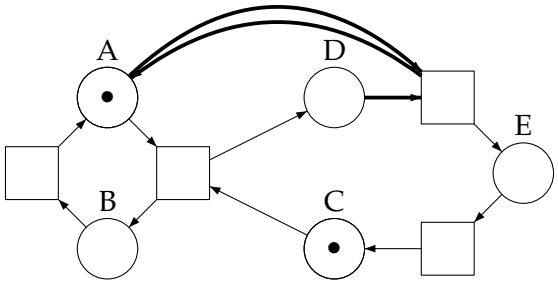


Figure 40: Past A, D

Odečtením této pasti od invariantu dostaneme

B - D $\leq$ 0

což vede k B $\leq$ D.

Podobným způsobem autor analyzuje varianty distribuovaných algoritmů vzájemného vyloučení. Tento způsob analýzy jsem použil v této diplomové práci v kapitole číslo 4.

První dvě kapitoly Analýzy základních systémových sítí pojednávaly o analýze statických vlastností. Tedy že určitý stav splňuje nějakou podmínku. Petriho sítě ale dokáží modelovat dynamický vývoj systému a právě analýzou vývoje se zabývá autor nyní. Autor analyzuje vývoj systému do určitého stavu a k tomu využívá analýzy sekvenčních a souběžných běhů.

V případě sekvenčního běhu zkoumá zda se v systému stav *a* vyvine do stavu *b*, či nikoliv. Tuto vlastnost nazývá "'vedoucí formulí"', která platí pokud pro každý stav s určitým indexem existuje v sekvenční běhu stav s indexem větším.

V jednoduché síti na obrázku 41 platí vedoucí formule A,B $\rightarrow$ D a neplatí formule A,B $\rightarrow$ A,E

Autor dále zobecňuje tyto vedoucí formule pro vlastnost, kterou nazývá *náchylnost k vývoji*. Náchylnost k vývoji je vlastnost stavu, ve kterém je umožněno provedení alespoň jednoho přechodu.

Další vlastností, kterou autor sleduje, je *zabránění přechodu*. V předchozím příkladu na obrázku 41 ve stavu BC provedením akce c zabrání provedení akce b. Všechny stavy, které nejsou zabráněny nějakým stavem, se nazývají *sadou změn* tohoto stavu. Tato sada změn stavu, který je tedy náchylný k vývoji, implikuje takzvanou *vedoucí formuli*.

Dohromady tyto vývojové vlastnosti autor spojuje do jediného formalismu, který nazývá *důkazový graf*. Tento acyklický graf, jehož uzly jsou tvořeny *stavovými formulemi*

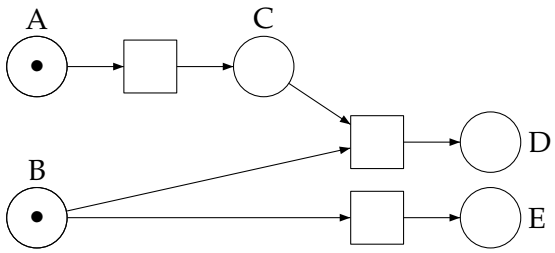


Figure 41: Vedoucí formule A, B $\rightarrow$ D

a orientované hrany disjunktní množinou vedoucích formulí, slouží k potvrzení výroků *stav a* $\rightarrow$ *stav b*.

Konstrukce takového grafu spočívá ve správném určení následných stavů z jediného vstupního stavu do jediného koncového stavu. Následný stav je určen na základě sady změn jednotlivých stavů s využitím jak invariantů, tak i pastí. Příklad takovéto konstrukce autor uvádí na straně 173 a 174 v [1]. Pro snazší pochopení tohoto postupu uvádím tento příklad také ve své diplomové práci.

Jedná se o dokázání vlastnosti vývoje ze stavu *připraven ke psaní* do samotného stavu *psaní*. Tyto stavy jsou reprezentovány místy B a C v asymetrickém algoritmu na obrázku číslo 42. V tomto algoritmu mají oba procesy různou prioritu pro vstup do své kritické sekce.[7]. Tedy o dokázání platnosti $\models$ B $\rightarrow$ C.

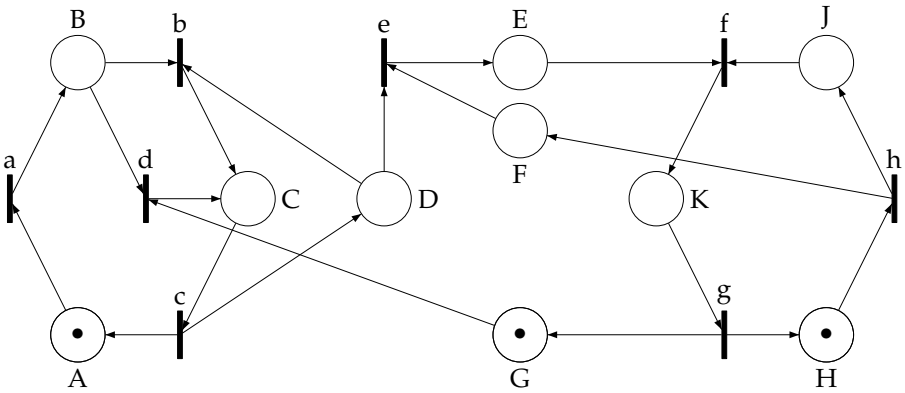


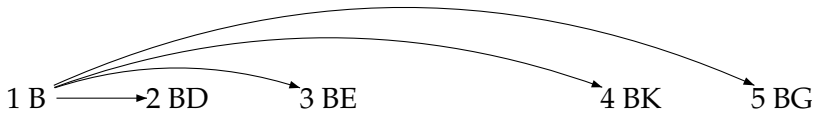
Figure 42: Asymetrický algoritmus vzájemného vyloučení

[7] Blíže je tento algoritmus popsán v kapitole 13.10 v [1]

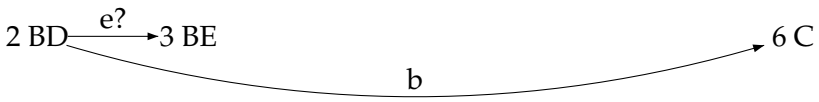
Protože stav B není sám *náchylný k vývoji* je nutné využít invariant D + E + K + G - A - B = 0 ze kterého vyplývá

$\models B \rightarrow (D \vee E \vee K \vee G)$

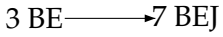
Z tohoto můžeme vytvořit $\models B \rightarrow (BD \vee BE \vee BK \vee BG)$



Prozkoumáme nové uzly. Uzel číslo 2 umožňuje akci *b*. Navíc z invariantu C + D + E + G + K = 1 vyplývá $\models D \rightarrow \neg G$ proto BD brání přechodu *d*, platí tedy $\models BD \rightarrow C \vee BE$



Otazník u přechodu *e* symbolizuje, že tento přechod nemusí být nutně umožněn. BE stejně jako první uzel neumožňuje nějakou akci, proto opět využijeme invariant, J - E - F = 0, který implikuje $\models E \rightarrow J$ a tedy i $\models BE \rightarrow BEJ$



Tento vytvořený uzel umožňuje provedení akce *f*, a zabraňuje akci *b* a *d*. Uzel BK umožňuje nápodobně pouze provedení akce *g*, vedoucí k BGH. Podobným způsobem prozkoumáme i uzel číslo 5 a vytvoříme kompletní důkazový graf algoritmu, který je uveden na obrázku číslo 43.

Takto představené důkazové grafy mají i své limity. V nespojitých sítích může dojít ke ztrátě informací a v případě některých formulí dokonce nejsou tyto grafy schopny potvrdit jinak platné tvrzení. V těchto příkladech je nutné síť rozšířit o "pomocné uzly" [8], umožňující provést analýzu důkazovým grafem.

Zavedením principu *fairness* do konstrukce důkazových grafů autor tuto kapitolu uzavírá a jak je u něj zvykem konstruuje v příkladových studií grafy dříve namodelovaných distribuovaných algoritmů. Autor se vrací ke všem algoritmům vzájemného

[8] například uzel znázorňující negaci jiného uzlu

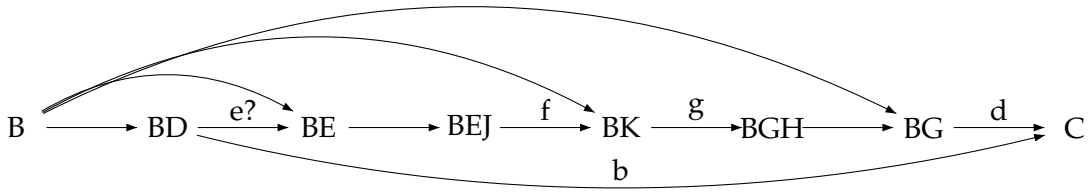


Figure 43: Důkazový graf $\models B \to C$

vyloučení popsaných v první části monografie a buď jejich důkazový graf přímo uvede, nebo čtenáře odkáže na místo, kde byl v rámci představení teorie již vytvořen.

Protože pouhý souvislý běh nedokáže vyjádřit všechny vývojové vlastnosti distribuovaných algoritmů, které často fungují na základě opakování akcí v několika kolech, Reisig čtenáři představuje souběžný vývoj distribuovaného algoritmu. Pro tento případ přináší nový operátor $\hookrightarrow$.

Podobně jako v předchozí kapitole byly definovány vedoucí formule, jsou v úvodu této části představeny "cause" formule v podobě $p \hookrightarrow q$. Tato formule je platná v souběžném běhu, pokud je každý dosažitelný p-stav následován dosažitelným q-stavem. Tyto causes formule jsou slabší než vedoucí formule a platí následující tvrzení:

Pokud $\models p \mapsto q$, pak platí i $\models p \hookrightarrow q$. Pokud q je disjunktní k progress prone množině stavů, pak pokud $\models p \hookrightarrow q$, tak platí i $\models p \mapsto q$.

Této vlastnosti můžeme v případě platné druhé podmínky využít pro důkaz platnosti vedoucích formulí při využití souběžného běhu.

Dále autor odvozuje vybírací pravidlo, pouze s tím rozdílem, že v případě souběžných jevů je lze přímo vkládat do kontextu. Například část souběžného běhu na obrázku číslo 44

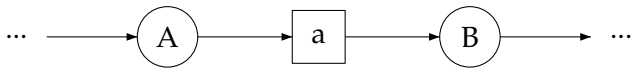


Figure 44: Souběžný běh $A \mapsto B$

představuje $\models A \mapsto B$, které lze rozšířit o jakýkoliv další stav C na $\models CA \mapsto CB$.

Takto definované causes formule se skládají pomocí výběrových pravidel do důkazového grafu, který je konstruován obdobným způsobem jako důkazový graf souvislého běhu.

Další prvek, který autor představuje k analýze souběžných běhů, nazývá ground formulí. Tou se nazývá dosažitelný stav následující každý jiný dosažitelný stav systému v každém jeho souběžném běhu.

Distribuované algoritmy pak mohou být pomocí těchto ground formulí zkoumány, porovnávány a dokazovány. Zajímavé ground formule jsou tvořeny množinou dosažitelných stavů a nazývají se ground stavy. Konečná část souběžného běhu, vycházejícího z ground stavu a také v něm končící, se nazývá "kruh" - kolo. Algoritmus, jehož všechny souběžné běhy lze popsat pomocí kol, se nazývá kruhový.

Pro přiblížení čtenáři autor tyto groud formule ukáže na algoritmech producent - konzument, které modeloval v předchozích kapitolách a ukáže právě rozdíl mezi ground formulí a ground stavem, tedy že algoritmus s ground formulí nemusí nutně obsahovat ground stav. Autor zde přednáší čtenáři prostý výčet kol, které jsou z jeho pohledu zajímavé. Sem patří například popis algoritmu hladových filozofů, který tvoří v originále pět kruhů, každý pro jednoho z filozofů, které se skládají z uchopení a položení hůlek.

Závěrem této části knihy autor analyzuje algoritmy vzájemného vyloučení, v nichž například provádí důkaz správnosti algoritmu, kdy ze stavu kdy první z procesů čeká a druhý je v kritické sekci, se systém vyvine do stavu, kdy je v kritické sekci proces první a proces druhý čeká.

5.5 Analýza pokročilých systémových modelů

Stavové vlastnosti v základních systémových modelech jsou definovány jako kombinace jejich lokálních stavů a jsou odvozeny od jejich platných rovnic a nerovnic. Ohodnocení jednotlivých míst rovnice představuje váhu jednotlivých míst ve určitém stavu. Pokud toto ohodnocení řeší systémovou rovnici nebo nerovnici, je tato rovnice či nerovnice platná v tomto stavu.

V případě rozšířených systémových modelů je tato váha tvořena nějakou doménou příslušející určitému místu. To znamená nutnost aplikovatelnosti funkce pro celý obsah místa v určitém stavu. Tento obsah autor pojmenovává *multimnožina*. Tyto multimnožiny potom slouží k vytváření rovnic a nerovnic, p-invariantů a iniciovaných pastí pokročilých systémových sítí.

Po tomto představení následuje formální definice stavových formulí analogicky k základním systémům a také definice již zmíněných multimnožin a definice jejich použití při vytváření stavových rovnic a nerovnic. Pro snazší přiblížení tématu čtenáři je uvedena příkladová studie algoritmu hladových filozofů.

Dále jsou definovány p-invarianty jako množina míst, ve které vážená množina odebraných tokenů se rovná vážené množině tokenů přidaných. Autor předvádí i stručnější reprezentaci pomocí mapování, které tvoří invariant tehdy, pokud je součet vah jednotlivých míst mapování roven nulové funkci.

Další strukturou, definovanou v této kapitole, jsou pasti. Autor i zde postupuje analogicky se základními systémovými sítěmi a pasti definuje jako množinu vah jednotlivých míst, kde pro jednotlivé elementy dané množiny platí: přechod, který odebere alespoň jeden token s váhou elementu z těchto míst, také vrátí alespoň jeden to-

ken s váhou elementu do těchto míst. Tento vztah lze, podobně jako v jednoduchých systémech, zapsat pomocí stavové nerovnice.

Závěr této kapitoly je věnován analýze běhů jak sekvenčních, tak souběžných. Sekvenční běh a jeho důkazové grafy se příliš neliší od těch z jednoduchých systémů, místo v síti nevyjadřuje pouze konkrétní místo, ale celou množinu míst. Důkazové grafy vytváří také analogicky k těm již dříve představeným. Následují formální definice z důvodu úplnosti definování popisovaného problému.

Analýza souběžného běhu také vychází z jednoduchých systémů a v zásadě se od nich neliší. Autor tuto část obohacuje o grafické znázornění možných šablon pro použití důkazových grafů.Mezi těmito šablonami nalezneme například synchronizační šablonu zobrazenou na následujícím obrázku.

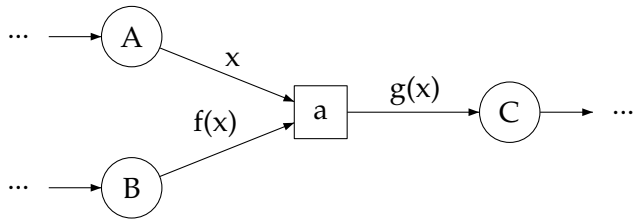


Figure 45: Souběžný běh A $\mapsto$ B

5.6 Formální analýza případových studií

Závěr celé monografie patří analýze případových studií pokročilých systémů, které nebyly analyzovány již v průběhu předchozího textu. Jednotlivé algoritmy jsou znovu reprezentovány v Petriho síti a autor, pomocí technik představených v kapitolách analýzy, dokazuje, že uvedené struktury skutečně mají ty vlastnosti, které se od nich vyžadují. Příkladem může být Echo algoritmus, kde autor dokazuje ukončení práce iniciátora až po informování všech míst a také, že iniciátor skutečně svoji činnost někdy ukončí. Čtenáři se tak dostanou praktické ukázky použití jednotlivých postupů.

5.7 Zhodnocení

Elements of Distributed Algorithms představují myšlenku využití formalismu Petriho sítí pro modelování a analýzu distribuovaných systémů. Tento, dříve nikde nepublikovaný přístup, přináší spojení grafové reprezentace s možností přímého modelování průběhu algoritmu.

Autor knihu nesměřuje k odborníkům na Petriho sítě nebo distribuované systémy, ale k širšímu okruhu čtenářů, kteří tak mohou těžit jak z celého teoretického základu

Petriho sítí, který je podrobně definován, tak i z výběru distribuovaných algoritmů, který pokrývá celé široké spektrum jejich použití.

Daní k tomuto přístupu je určitá povrchnost, kdy jsou jednotlivé algoritmy předkládány bez užších souvislostí nebo bez důvodu jejich výběru. Autor sám přiznává, že není jeho cílem ponoření do detailů a že každý sebesložitější model musel mít svůj jednoduchý základ. Tento základ se snaží popsat zároveň se základními pravidly pro jeho rozšíření.

Díky tomuto relativně jednoduchému přístupu lze tuto knihu využít jako základ pro studium Petriho sítí, který je představen na aktuálních výpočetních problémech spojených s distribuovanými algoritmy. Monografie může dále sloužit jako představení problematiky distribuovaných algoritmů spolu s prostředky jejich modelování a analýzy.

Použití této monografie pro vytváření modelu konkrétního komplexního problému je možné pouze v počátečních fázích vývoje. Petriho sítě představené v této knize nejsou schopné popsat složitější procesy, a proto je nutné je rozšířit nad rámec zde popsaný. Pro tento druh práce je vhodnější například monografie Coloured Petri Nets (Springer 2009)[11] díky svému praktickému zaměření. [9]

[9] Srovnání s touto knihou nabízím v příloze diplomové práce

6 Animace

6.1 Úvod

Praktickým úkolem této diplomové práce je vytvoření didakticky vedených animací pro znázornění výhody reprezentace Petriho sítí pro modelování a analýzu distribuovaných algoritmů. V této kapitole se čtenář seznámí s vlastním procesem tvorby těchto animací a také s jejich stručným popisem.

6.2 Proces tvorby

Proces vytvoření elektronických animací pro tuto diplomovou práci prošel několika vývojovými stupni. Jako první řešení jsem zvolil elektronickou prezentaci pomocí nástroje Beamer. Tento nástroj je rozšiřující třída sázecího systému LaTeX a umožňuje vytváření prezentací. Díky vytvořeným modelům distribuovaných algoritmů v rámci diplomové práce v systému LaTeX bylo jednoduché tyto struktury přenést do systému Beamer a pouze pomocí dílčích změn ve struktuře přivést celý systém k životu.

Struktura vytvořené prezentace se skládala vždy z představení algoritmu vzájemného vyloučení, nástinu průběhu tohoto algoritmu a provedení kroků, které vedou k určitému, didakticky zajímavému stavu nebo problému, který je následně zanalyzován. Animace byla vystřídána formálním definováním dalších struktur, které tento problém řeší, a opět představením příslušného modelu. Po dohodě s vedoucím diplomové práce jsem od tohoto konceptu upustil s tím, že veškeré formální definice jsou již obsaženy v textu diplomové práce a je žádoucí se více věnovat přímo jednotlivým modelům a jejich analýze.

Po vypuštění pouze textových částí se ukázalo mnohem složitější připojení doprovodného textu přímo k vytvořeným modelům. Tento nedostatek částečně spočívá v samotném nástroji Beamer, proto jsem se ho snažil nahradit aplikací MS Office Powerpoint 2003. Ani tady jsem nebyl s výstupem spokojený. Výsledkem tohoto přístupu se stalo několik prezentací, rozdělených na samostatné modelování a samostatný popis. Uživatel by tak musel neustále přepínat mezi prezentacemi a nemohl by se tak soustředit na vlastní obsah.

Základ, použitý pro vytvoření finálních animací, mi poskytnul nástroj CPN Tools. V jeho prostředí jsem již dříve vytvořil řadu modelů, popisujících jak různé možnosti modelování v tomto nástroji, tak i dokumentující rozdíl mezi různými nástroji pro vytváření modelů. Tyto skripty také tvoří přílohu této diplomové práce a mohou sloužit pro snazší proniknutí do problematiky Petriho sítí a distribuovaných algoritmů.

Jednotlivé modely jsou nasnímány do avi souborů, ke kterým jsem vytvořil odpovídající otitulkování.

6.3 Popis jednotlivých animací

Tématem těchto animací je algoritmus vzájemného vyloučení, tak jak jsem ho modeloval a analyzoval v rámci kapitol 3 a 4.

Vzájemné vyloučení 1 zobrazuje nejjednodušší model algoritmu vzájemného vyloučení. V rámci animace je předvedena struktura jeho Petriho sítě a znázorněn průběh vlastního algoritmu. V závěru je představena jeho analýza pomocí stavového prostoru. V rámci analýzy stavového prostoru je provedena i kontrola vlastnosti vzájemného vyloučení a potvrzena jeho správnost.

Vzájemné vyloučení 2 zobrazuje algoritmus využívající proměnnou tah. V rámci animace je předveden průběh algoritmu s ohledem na sekvenční střídání jednotlivých procesů. V tomto algoritmu je zdůrazněn problém zastavení jednoho z procesů, který způsobí zastavení celého systému. Dále je zobrazena jeho analýza pomocí stavového prostoru. V něm je provedena i kontrola vlastnosti vzájemného vyloučení a potvrzena jeho správnost.

Vzájemné vyloučení 3 zobrazuje algoritmus využívající kontrolní proměnné pro vstup do kritické sekce. Tyto proměnné mají signalizovat, zda se jednotlivé procesy chystají vstoupit do kritické sekce, a případně zamezit ostatním procesů ve vstupu do svých kritických sekcí. V rámci animace je předveden průběh algoritmu a jeho analýza pomocí stavového prostoru. V něm je provedena i kontrola vlastnosti vzájemného vyloučení, která vede ke zjištění porušení základní podmínky tohoto algoritmu. Tato situace je následně přímo modelována i ve struktuře Petriho sítě.

Vzájemné vyloučení 4 zobrazuje algoritmus využívající kontrolní proměnné pro vstup do kritické sekce s ohledem na neplatný algoritmus z předchozího příkladu. V rámci animace je předveden průběh algoritmu a jeho analýza pomocí stavového prostoru. V něm je znázorněno uzamčení v tomto systému, které je ze stavového prostoru jasně patrné. Tato situace je následně modelována i ve struktuře Petriho sítě.

Vzájemné vyloučení 5 zobrazuje algoritmus využívající kontrolní proměnné pro vstup do kritické sekce s ohledem na neplatné algoritmy z předchozích příkladů. V rámci animace je předveden průběh algoritmu a jeho analýza pomocí stavového prostoru. V něm je provedena i kontrola vlastnosti vzájemného vyloučení a potvrzena jeho správnost.

Dekkerův algoritmus zobrazuje algoritmus využívající kontrolní proměnné pro vstup do kritické sekce a dále proměnnou tah. Je naznačena podobnost s předchozími modely. V rámci animace je předveden průběh algoritmu a jeho analýza pomocí stavového prostoru. V té je demonstrována rozsáhlost stavových prostorů ve složitějších systémech.

6.4 Závěr

Využití nástroje CPN Tools a v něm vytvořených modelů přináší široké možnosti použití v rámci výuky. Pedagog může využít mnou připravené animace tak, jak jsou, případně si může upravit texty jednotlivých titulků. V případě potřeby si však dokáže vytvořit pomocí připravených skriptů a nástroj, obsažených v příloze animace úplně vlastní, které lépe postihnou jeho potřeby.

7 Závěr

Hlavní cílem mé diplomové práce bylo prozkoumat možnosti modelování a analýzy distribuovaných algoritmů pomocí grafové reprezentace Petriho sítěmi. S ohledem na tento cíl jsem spolu vedoucím diplomové práce vybral několik algoritmů, řešících typické distribuované problémy. Představení těchto algoritmů a jejich standardní zápis v pseudoprogramovém kódu doplňují i grafické reprezentace pomocí Petriho sítí a čtenáře navíc postupně seznamují s jejich teorií. Uvedené modely ukazují výhody takovéto reprezentace, která je snadno pochopitelná a umožňuje kontrolu průběhu algoritmu přímo v simulaci modelu. Všechny Petriho sítě uvedené v mé diplomové práci jsem vytvářel pomocí nástroje gastex a díky své struktuře jsou připraveny pro pozdější případné strojové zpracování.

Druhým dílčím úkolem bylo jeden z algoritmů podrobit podrobnému modelování a analýze. Pro tento úkol jsem zvolil algoritmus vzájemného vyloučení, zejména z důvodu možnosti porovnání vývoje s monografií Elemements of Distributed algorithms. Postupné rozšiřování tohoto algoritmu jsem popsal řadou kroků, kdy každý pokročilejší stupeň odstranil nějaký nedostatek, vyskytující se ve stupni předchozím. I zde je jasně patrná výhoda reprezentace pomocí Petriho sítí, kdy v jednotlivých modelech lze snadněji identifikovat problematická místa bez potřeby dalších dodatečných informací. Modely vyvinuté pro tuto diplomovou práci jsem posléze porovnal s modely uvedenými W. Reisigem. V počáteční fázi vývoje jsou modely identické, ale jak systém mohutní, objevují se rozdílnosti. Nejvíce je to patrné u Dekkerova algoritmu, který v mé verzi více odpovídá popisu uvedeném v [6]. Tyto rozdíly jsem popsal jak v kapitole vzájemného vyloučení, tak i v kapitole věnující se obsahu Reisigovy monografie.

Pro potřeby analýzy Petriho sítí jsem čtenářům představil dvě možné metody analýzy - algebraickou a grafovou. Obě tyto metody jsem použil na analýzu modelů distribuovaných algoritmů vzájemného vyloučení. Mým cílem bylo zejména dokázat jejich základní vlastnost, tedy v případě vstupu jednoho z procesů do své kritické sekce, druhý z procesů do své kritické sekce vstoupit nemůže. Tato analýza zároveň potvrdila správnost mnou navržených modelů.

Posledním teoretickým úkolem bylo seznámit čtenáře této diplomové práce s monografií W. Reisiga Elements of Distributed Algorithms. Na základě popisu vytváření modelů distribuovaných algoritmů pomocí Petriho sítí jsem z této knihy vycházel také při vytváření modelů pro svou diplomovou práci. Mohu tedy přímo posoudit výhody a nevýhody této monografie při plnění tohoto úkolu. V počáteční fázi vývoje systému lze tuto knihu doporučit, protože čtenáře seznamuje jak se základy modelování a analýzy algoritmů, tak i s teorií potřebnou pro pochopení Petriho sítí. Bohužel jak se systém vyvine do větší šířky nebo hloubky, je problematické Petriho sítě, tak jak je navrhuje Reisig, použít. Důvodem je jeho snaha o co největší jednoduchost modelu, což při složitých algoritmech není možné použít. Další nevýhodou této monografie je nepřítomnost programové aplikace, ve které by si mohl čtenář probíranou problematiku prakticky vyzkoušet.

Praktickým úkolem této diplomové práce bylo vytvoření didaktických animací pro popis vytvořených modelů. Pro tyto účely jsem vytvořil sadu animací algoritmu vzájem-

ného vyloučení, ve kterých jsou jednotlivé algoritmu popsány a analyzovány. Součástí každé animace je představení některého z problémů, které mohou v tomto algoritmu nastat. Jako základ těchto animací slouží sada skriptů vytvořených v aplikaci CPN Tools, které sami o sobě mohou také sloužit budoucím čtenářům pro přiblížení algoritmů vzájemného vyloučení. Z důvodu porovnání různých existujících nástrojů pro vytváření Petriho sítí jsem vytvořil obdobnou sadu skriptů i v nástroji jPNS.

Práce na tomto projektu mi přiblížila problematiku Petriho sítí, jejich teorii i různé nástroje pro jejich modelování a analýzu, a to nejen nástroje pro vlastní modelování, jako je CPN Tools, ale také sadu maker gastex, která mi umožnila vytvářet tyto modely v prostředí editoru LaTeX. Dále jsem se seznámil s několika distribuovanými algoritmy a naučil jsem se vytvářet jejich model jako Petriho síť. Díky této práci jsem se také seznámil s monografií Elements of Distributed algorithms a Coloured Petri nets, které pomocí stručných popisů přibližuji i dalším čtenářům.

Práce s nástrojem CPN Tools mi přinesla řadu poznatků a zkušeností, které hodlám uplatnit ve své stávající praxi. Pomocí tohoto nástroje se pokusím vytvořit hiearchický model firmy, ve které jsem zaměstnán, jehož uzly budou představovat jednotlivá oddělení. Tato oddělení pak budou opět popsána pomocí barevných Petriho sítí, s možností simulace jednotlivých zaměstnaců nebo datových vstupů a výstupů. Pomocí CPN Tools pak bude možné navíc představit dynamiku systému nebo procesy zkontrolovat pomocí stavových diagramů. V tomto projektu budou využity i poznatky z oblasti distribuovaných algoritmů, zejména řešení vzájemného vyloučení nebo sdílených zdrojů.

A Literatura

[1] Reisig, W.: *Elements of distributed algoritms*. Springer, 1998. ISBN 3-540-62752-9

[2] Lynch, N. A.: *Distributed algorithms*. Morgan Kaufmann Publishers Inc., 1996. ISBN 978-1-55860-348-6

[3] Santoro, N. : *Design and analysis of distributed algorithms*. John Wiley & Sons, Inc., 2007. ISBN 978-0471719977

[4] Kochaníčková, M.: *Petriho sítě*. Univerzita Palackého, Olomouc, 2008

[5] Dijkstra, E. W.: *Hierarchical Ordening of Sequential Processes*[online][cit. 2010-12-31]. Dostupné z `http://www.cs.utexas.edu/users/EWD/ewd03xx/EWD310.PDF`

[6] Dijkstra, E. W.: *Cooperating Sequential Processes*[online][cit. 2010-12-31]. Dostupné z `http://userweb.cs.utexas.edu/users/EWD/ewd01xx/EWD123.PDF`

[7] Lynch, N. - Saias, I. - Segala, R.: *Proving Time Bounds for Randomized Distributed Algorithms*[online][cit. 2010-12-31]. Dostupné z `http://arxiv.org/PS_cache/math/pdf/9409/9409221v1.pdf`

[8] Garcia-Molina, H.: Elections in a Distributed Computing System. *IEEE Transactions on Computers*. January 1982, Vol. C-31, No. 1, s. 48-59.[online][cit. 2010-3-16]. Dostupné z `http://paul.rutgers.edu/cs545/S02/papers/molina-elections.pdf`

[9] Chang, E. - Roberts, R. An improved algorithm for decentralized extrema-finding in circular configurations of processes. *Communications of the ACM*. May 1979, Vol. 22, No. 5, s. 281-283.[online][cit. 2010-3-16]. Dostupné z `http://compalg.inf.elte.hu/~tony/Oktatas/Osztott-algoritmusok/p281-chang.pdf`

[10] Gallagher, R. G. - Humblet, P. A. - Spira, P. M.: A Distributed Algorithm for Minimum-Weight Spanning Trees. *ACM Transactions on Programming Languages and Systems*. January 1983, Vol. 5, No. 1, s. 66-77.[online][cit. 2010-3-20]. Dostupné z `http://www.cs.tau.ac.il/~afek/p66-gallager.pdf`

[11] Jensen K. - Kristensen L. M.: *Coloured Petri nets*. Springer, 2009. ISBN 978-3-642-00283-0

B jPNS

B.1 Úvod

Tato kapitola se zabývá volně dostupným nástrojem jPNS, který slouží pro modelování Petriho sítí. Jako přílohu k této diplomové práci jsem vytvořil sadu skriptů, které simulují jednotlivé distribuované algoritmy zmiňované v mé diplomové práci. Autorem tohoto simulátoru Petriho sítí je Thomas Bräunl. V tomto simulátoru je možné vytvořit i poměrně složité Petriho sítě a simulovat jejich průběh v sekvenčním nebo paralelním režimu.

B.2 Instalace a spuštění

Aplikaci je možno spustit přímo online z adresy (bez možnosti uložení na místní disk) `http://www.informatik.uni-hamburg.de`.

Druhou možností je stáhnutí celé aplikace a lokální instalace. Aplikaci lze stahovat z adresy `http://robotics.ee.uwa.edu.au/pns/ftp/`.

jPNS je možné spouštět jak ze systémů Linux, tak ze systémů MS Windows.

Po rozbalení a uložení archivů aplikaci spustíme příkazem "java jpns" v adresáři, kam jsme aplikaci instalovali.

Popis ovládacích prvků této aplikace je možné nalézt v nápovědě, která se skrývá pod volbou HELP.

B.3 Ovládání

Pro začátek práce s jednoduchými, bezpečnými Petriho sítěmi je tento program dostatečně vybavený všemi potřebnými funkcemi. Vlastní vytváření jednotlivých diagramů je intuitivní a přehledné. Za nevýhody tohoto programu lze považovat zejména absence jakékoliv analýzy průběhu algoritmu, a proto není možné automaticky ověřit požadované vlastnosti. S tím souvisí i nemožnost vytvoření nějakého scénáře průběhu algoritmu, na kterém by bylo možné simulovat nějaký konkrétní speciální stav. Verifikace jednotlivých algoritmů pak v praxi znamená pokusit se manuálně projít krok za krokem jednotlivé dosažitelné stavy, a tak potvrdit zda algoritmus funguje, jak má.

Z hlediska grafického výstupu a funkčních možností lze kladně hodnotit možnost jednoduchého pojmenování všech uzlů, přiřazení priorit do přechodů a možnost nastavení zvýraznění toho přechodu, který je právě procházen. Naopak chybí možnost popisu hran. Největším problémem však zůstává nemožnost vytvoření hrany jako křivky a tím pádem i ke značně nepřehlednému křížení jednotlivých hran v případě složitějších algoritmů.

Součástí této diplomové práce jsou i v tomto programu vytvořené skripty vybraných základních algoritmů, které tato práce popisuje, ale nejsou obsaženy v elektronické animaci. Tyto skripty mají sloužit zejména těm čtenářům, pro které je problematika Petriho sítí úplně neznámá. Tuto cestu jsem zvolil zejména z toho důvodu, že prostá simulace

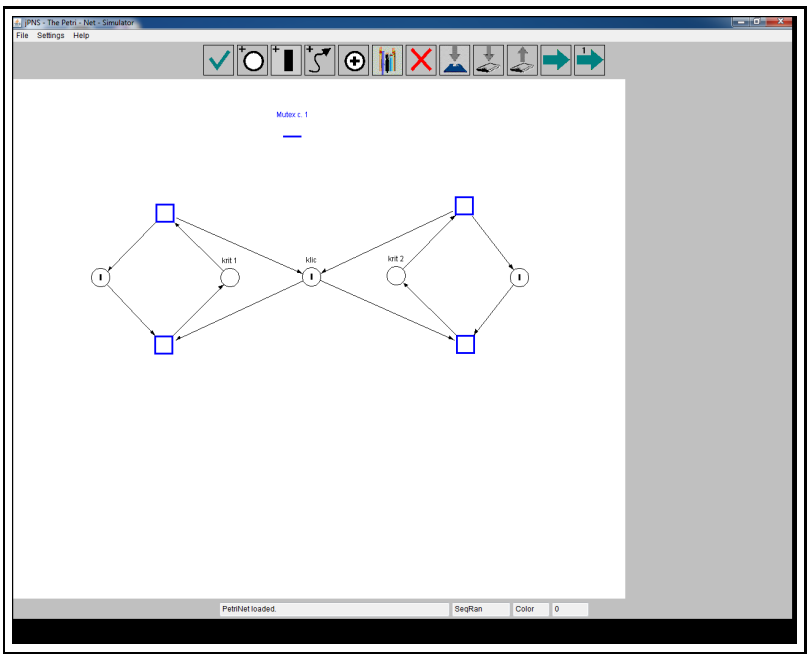


Figure 46: Pracovní prostředí aplikace jPNS

jednotlivých algoritmů není předmětem mé práce a v rámci elektronické animace se chci spíše věnovat simulaci různých zajímavých stavů těchto algoritmů.

C CPN Tools

C.1 Úvod

Tato kapitola se zabývá jiným opět volně dostupným nástrojem CPN TOOLS, který také slouží pro modelování Petriho sítí. Stejně jako s předchozím nástrojem jsem vytvořil sadu skriptů, které simulují jednotlivé distribuované algoritmy zmiňované v mé diplomové práci a přikládám je jako přílohu. Tento nástroj je již poměrně složitější, než byl jPNS a umožňuje vytvářet barevné, hierarchické Petriho sítě a také provádět jejich základní analýzu. Dalším důvodem pro výběr tohoto nástroje je srovnání přístupu monografie Coloured Petri Nets (Springer 2009)[11], ze které CPN Tools vycházejí, s monografií Elements of distributed algoritms W. Reisiga [1].

C.2 Instalace a spuštění

CPN TOOLS lze stáhnout z adresy `http://cpntools.org`, která obsahuje všechny informace o tomto projektu spolu s podrobným návodem instalace a spuštění. Nápovědu lze zobrazit ze stránky `http://cpntools.org/gettingstarted/start`

C.3 Monografie Coloured Petri nets

Tato monografie[11] je aktualizovaným souhrnem informací a poznatků obsažených v původní třídílné podobě, vydávané K. Jensenem v letech 1992 - 1997, a zhodnocuje v sobě informace a zkušenosti z více jak desetiletého používání barevných Petriho sítí jejich autory při praxi i výuce. Monografie představuje využití barevných Petriho sítí pro modelování a analýzu souběžných systémů jak po teoretické stránce, tak i v praktických příkladech.

Zaměření této monografie je odlišné od pojetí W. Reisiga v Elements of Distibuted algorithms. Závěrem této kapitoly bych chtěl oba tyto přístupy vzájemně porovnat.

Barevné Petriho sítě (CPN) jsou nástrojem grafického modelování souběžných systémů a jejich analýzy s využitím grafické reprezentace spojené s programovacím jazykem. CPN ML je modelovací jazyk, umožňující definování datových typů, popisování manipulace dat a vytváření měřitelných modelů. Jeho zaměření není úzce specializované a umožňuje proto zpracovávat široké spektrum modelů a situací. Tato monografie je podpořena softwarovým nástrojem - CPN Tools, který využívá tohoto přístupu spojení grafové reprezentace s možnostmi programového jazyka.

Posloupnost kapitol poukazuje spíše na praktické využití této knihy. Autoři přecházejí od konkrétních příkladů k jejich teoretickému základu v podstatně větší míře než W. Reisig ve své monografii.

V úvodu jsou představeny základní prvky konstrukce Barevných Petriho sítí na modelu komunikačního protokolu. Tento model čtenáře doprovází po celé knize a je na něm demonstrována většina modelových situací, včetně neformálních základů teorie Petriho sítí. Tato kapitola nevyžaduje žádnou zkušenost s touto tématikou a měla by být srozumitelná pro většinu čtenářů.

V následující kapitole je představen CPN ML programovací jazyk, který již byl zmíněn v úvodu a v této kapitole je podrobněji popsán. Tento jazyk slouží k definování množin barev, datových typů a funkcí i k výpočetním a dotazovacím funkcím.

Kapitolu formálních definicí Barevných Petriho sítí lze při prvním čtení vynechat a vycházet z neformálních popisů v předchozích kapitolách.

V následující části jsou popsány techniky vytvoření hierarchických Barevných Petriho sítí z jednotlivých modulů spolu s jejich výhodami, reprezentovanými opakovaným použitím některých částí struktury nebo z důvodu zjednodušení příliš rozvětveného systému.

I tato část je následována formální definicí struktur v ní popsaných, které však mohou být při prvních čteních také přeskočeny.

Pro potřeby analýzy následuje kapitola popisující stavový prostor jednotlivých modelů a jeho ověřování. Jednou z funkcí aplikace CPN Tools je vytvoření stavového prostoru vytvořeného modelu a jeho následná grafická reprezentace. Pomocí těchto nástrojů je možné automaticky ověřit některé vlastnosti modelu, jako je dosažitelnost míst nebo maximální a minimální počet tokenů v určitém místě. Samozřejmou součástí je formální definování stavového prostoru jako takového.

Použití CPN ML modelovacího jazyka umožňuje vkládání časového prvku do struktury Petriho sítí. Tato kapitola popisuje využití tohoto časového prvku pro analýzy efektivnosti průběhu operací systému i pro analýzu systémů v reálném čase. Opět je tato kapitola následována formálním definováním časových barevných Petriho sítí.

Závěr monografie tvoří využití tohoto formalismu v konkrétních projektech. Výběr projektů zohledňuje široké využití CPN Tools pro různé stupně vývoje projektů jak v soukromých tak i ve státních organizacích. Některé z těchto projektů jsou v této kapitole blíže prozkoumány, na jiné jsou uvedeny reference.

Poslední kapitola shrnuje použití této monografie spolu s nástrojem CPN Tools při výuce kurzu Modelování a verifikace souběžných systémů na univerzitě v Aarhusu. Kapitola zahrnuje jak studijní plán tak i očekávané výsledky studia a zároveň popisuje praktické příklady vhodné pro použití lektorem.

C.4 Popis aplikace CPN Tools

CPN Tools je nástroj pro editaci, simulaci, analýzu stavového prostoru a analýzu efektivnosti modelů vytvořených pomocí Barevných Petriho sítí. Podporuje hierarchické i nehierarchické, časované i nečasované Petriho sítě. V současné době jej používá více jak 8000 uživatelů v mnoha zemích. Nástroj je možné používat jak v Linuxu, tak i ve Windows XP a vyšších.

Vytváření a analýza modelu se provádí přímo prostřednictvím grafického rozhraní, které obsahuje interaktivní palety a výběry, jak je možné vidět na následujícím obrázku.

Obrazovka je rozdělena na tzv. *workplace* a *index*. Workplace - pracovní plocha je místo, ve kterém uživatel vytváří grafický model systému. Na této pracovní ploše může mít otevřeno několik oken - *binders*. Ty obsahují buď přímo prvky vytvářeného modelu, nebo nástroje, sloužící například k simulaci systému. Stiskem pravého tlačítka myši na jednotlivých prvcích pracovní plochy jsou vyvolávány kruhové kontextové nabídky.

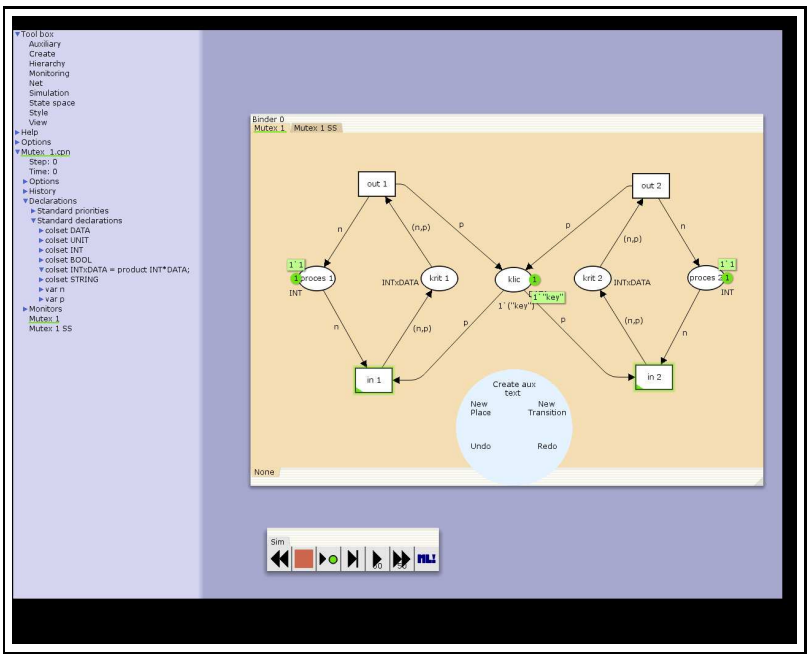


Figure 47: CPN Tools

V levé části obrazovky se nachází tzv. index, který obsahuje jak přednastavené palety nástrojů, tak i sekce, sloužící pro uživatelské nastavení. Sem patří zejména sekce *deklarací*, ve které musí uživatel deklarovat nové barevné množiny nebo proměnné.

Postup vytvoření modelu a jeho jednoduchou analýzu jsem zpracoval v prezentaci, která je součástí této diplomové práce. Pomocí této prezentace chci budoucím zájemcům přiblížit krok za krokem postup vytvoření jednoduchého modelu a jeho analýzu. Tento nástroj sice obsahuje velkou a interaktivní stránku podpory, ale ta kopíruje sekcemi obsah monografie. To znamená, že prvními lekcemi není vytvoření modelu, ale pouze pozorování již připravených příkladů. Pokud tyto příklady uživateli nevyhovují, musí postupně procházet další sekce podpory a snažit se najít postup pro jednotlivé kroky vytvoření vlastního modelu.

Proto má prezentace sleduje vývoj modelování v krocích od vytvoření uzlů, definování jejich barev, propojení uzlů pomocí hran a deklarací proměnných jednotlivých hran. V postupu jsou zohledněny podmínky pro korektní analýzu, vyplývající z mé práce s nástrojem. Případní zájemci tak nemusí ztrácet tolik času hledáním správného postupu, ale mohou se věnovat přímo tvorbě modelu a jeho následné analýze.

C.5 Srovnání s Elements of distributed algoritms

Prostudování monografie Coloured Petri nets mi přináší možnost srovnat přístup jejich autorů s přístupem W. Reisiga v Elements of Distributed Algorithms. Obě knihy popisují využití Petriho sítí k modelování a analýze distribuovaných systémů s využitím grafové

reprezentace. Přestože teoretická část obou knih je víceméně stejná, celkové zaměření se odlišuje.

W. Reisig k tomuto tématu přistupuje více po teoretické stránce. Členění jeho práce vždy postupuje od teoretického základu k praktickým příkladovým studiím. Autor se nesnaží jít příliš do hloubky a předkládá čtenáři postupně řadu modelů distribuovaných algoritmů. Při jejich popisu a analýze vede čtenáře různými teoretickými scénáři vývoje systému tak, aby poukázal na důležité situace nebo problémy. Coloured Petri nets naproti tomu provází čtenáře od praktických pozorování jediného algoritmu, v mírných úpravách, k jejich teoretické definici. Čtenář má tak možnost neformálně pochopit všechny popisované problémy a pokud se zajímá i o jejich formální definici, má ji možnost nastudovat později. Navíc popis pomocí jediného algoritmu pomáhá díky ucelenosti snáze pochopit jednotlivé části teorie.

Vzhledem k použitým prostředkům se také odlišují metody analýzy v jednotlivých monografiích. Použitím jednoduchých Petriho sítí umožňuje W. Reisigovi analýzu pomocí algebraických metod, zejména pomocí pastí a invariantů. Tyto speciální struktury však nejsou popsány v monografii Coloured Petri nets a jejich vyhledání a analýza není implementována ani v CPN Tools. Důvodem této situace je nemožnost, zejména pro rozsáhlé systémy, těmito invarianty pokrýt celou síť, tedy není možné je všechny vyhledat. Právě rozsáhlé Petriho sítě mohou dosahovat stavů systému v řádů tisíců nebo více.

Na druhou stranu se W. Reisig, v rámci zachování relativně jednoduchých systémů, nevěnuje Petriho sítím s časovým prvkem. Takto upravené Petriho sítě obsahují dodatečné informace o čase, ve kterém jsou například umožněny přechody. Přidání tohoto časového aspektu pak umožňuje analýzu systému nejen z hlediska správné funkce, ale umožňuje analyzovat i například časovou náročnost systému nebo některé jeho části.

Velkou výhodou Coloured Petri nets je jejich provázání s programovým nástrojem CPN Tools. Toto spojení přináší budoucím zájemcům jedinečnou možnost si jednotlivé problémy vyzkoušet v tomto nástroji a využít tak jeho modelovacích a analytických funkcí. W. Reisig bohužel nepřináší ve svém díle takovou možnost a ani na jiný nástroj nepoukazuje. Čtenář jeho monografie je tak odkázán na nástroje, které plně nekorespondují s jeho představou a nemají tak možnost úplné simulace jeho modelů.

Celkově je monografie Coloured Petri nets určena pro prakticky zaměřené čtenáře, kteří chtějí tuto knihu a z ní vycházející nástroj použít pro řešení konkrétních praktických úkolů, bez potřeby hlubšího ponoření do teoretických definicí.

D Programové prostředky použité pro vytvoření animací

Pro vytvoření animací, které jsou součástí této diplomové práce, byly použity následující programové prostředky:

- **CPN Tools** dostupné z `http://cpntools.org/`
- **CamStudio** dostupné z `http://camstudio.org/`
- **VirtualDub** dostupné z `http://www.virtualdub.org/`
- **Subtitle Workshop** dostupné z `http://www.urusoft.net/`